TU WIEN Informatics

Streaming and Quantitative Extensions of Answer Set Programming

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

MSc. Rafael Kiesel
Registration Number 11840527

to the Faculty of Informatics

at the TU Wien

Advisor: O.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Second advisor: Univ.Prof. Dr. Ezio Bartocci

The dissertation has been reviewed by:

Fabrizio Riguzzi	Pedro Cabalar

Vienna, 1st September, 2023

Rafael Kiesel

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

MSc. Rafael Kiesel

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. September 2023

Rafael Kiesel

The natural numbers? That semi-ring's a bell.

Acknowledgements

First and foremost, I would like to express great thanks to my supervisor Thomas Eiter. He was and is everything one could hope for in a mentor. Not only did he take all the time needed to have in depth discuss and guide me in my research - he at the same time left me the freedom to study, learn, and dive deep into topics that interested me. With this self-study and discussion feedback loop he patiently both taught me what constitutes good research and lead me to develop an understanding which stones should be turned towards it and which to leave unturned so as not to get stuck under them. In short, I am very grateful for the extraordinarily pleasant and fruitful experience of his supervision.

Furthermore, I want to thank Fabrizio Riguzzi and Pedro Cabalar for the many insightful comments that helped me improve this thesis.

My experience as a PhD student was further delighted by a vast number of people:

- Alexandra Pavlova, for lighting up my days no matter how cloudy, with inspiring conversations, stupid jokes, and an open ear for even the least interesting of my stories.
- André Schidler, for his company during breaks and the invitation into the world of scientific programming challenges. His incredible dedication and insightful discussions paired with his chill personality made me realize how much fun playing with daggers can be.
- Markus Hecher, for making work meetings feel like a hangout among friends that just casually lead to cool results.
- Anna Rapberger, for teaching me Tarock, accepting me even though I keep on saying “laufen”, and the countless hours spent during breaks.
- Sanja Lukumbuzya, for the deep conversations over lunch, the joy she radiates, when she’s in the room, and the countless hours spent during breaks.
- Kees van Berkel, for amazing conversations, where nothing is serious but also for serious conversations with advice that is amazing.
- Thekla Hamm, for letting us move her stuff to Germany.

- Peter Skocovsky, for being an even greater friend than coworker.
- David Cerna, for inviting me to the first and most cited paper of my PhD.
- My short-term colleagues in Belgium, for welcoming me with open arms.
- Kilian Rückschloßand Felix Weitkämper, for the productive collaboration that felt more like friendship.
- Anna Prianichnikova, for caring not only for me but the whole LogiCS DK, as well as hopefully soon to be repeated conversations over lunch.
- All the other members of the LogiCS DK that these acknowledgments are way to short to mention in the manner they deserve.
- My friends in Germany, with whom a few days of moving are more fun than weeks of vacation anywhere else. I have more pleasant memories with them than there are words in this thesis.
- My family, who are like family to me.
- Rafael Kiesel, without whom I would have never managed to finish this thesis.

Abstract

Stream reasoning allows us to draw conclusions in a temporal domain with data that can change at different time points. By using a stream reasoning framework with syntax and semantics based on that of Answer Set Programming (ASP), one obtains an intuitive and declarative formalism.

However, while there are vast possibilities to perform *quantitative reasoning* over static data, the same does not hold in the streaming context. Additionally, (i) there are many different forms quantitative reasoning that (ii) are not trivially integrated into to the temporal setting.We therefore investigate the combination of streaming and quantitative extensions of ASP.

First, we introduce two highly general quantitative extensions of ASP by defining an *algebraic semantics* for quantitative aspects that is based on *semirings*. These extensions add on the one hand quantitative reasoning capabilities over the set of models and on the other hand succinct specifications of quantitative constraints within the program. For both, we analyze their relation to previous formalisms with similar features mostly showing that ours are a conservative extension. Finally, we combine the quantitative extensions with a temporal framework called LARS to obtain a general framework for quantitative stream reasoning.

Apart from their definition, we study the extensions and the general framework in terms of their theoretical properties, including the complexity of typical reasoning tasks, safety of fragments, and expressivity.

Second, we notice that reasoning in both extensions involves solving weighted model counting problems over semirings. Interestingly, the complexity of the problem depends on the semiring. However, this dependence has not been characterized. We provide a characterization using a family of novel complexity classes and relate them to well-known classical complexity classes.

Last but not least, we consider how reasoning in practice. For this, we only consider a limited fragment, omitting quantitative constraints and temporal aspects but focusing on general quantitative reasoning over the set of models. By a mixture of known results from the literature and novel findings we provide an implementation that at least keeps up with the state of the art in probabilistic reasoning and even provides improved performance on cyclic instances.

TU WIEN Informatics

Zeitliche und Quantitative Erweiterungen der Answer Set Programmierung

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

MSc. Rafael Kiesel
Matrikelnummer 11840527

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: O.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Eiter
Zweitbetreuung: Univ.Prof. Dr. Ezio Bartocci

Diese Dissertation haben begutachtet:

Fabrizio Riguzzi	Pedro Cabalar

Wien, 1. September 2023

Rafael Kiesel

Kurzfassung

Stream Reasoning ermöglicht es uns, Schlussfolgerungen in einem zeitlichen Bereich mit Daten zu ziehen, die sich zu verschiedenen Zeitpunkten ändern können. Durch die Verwendung eines Stream-Reasoning-Frameworks mit Answer Set Programmierung (ASP)-basierter Syntax und Semantik, erhält man einen intuitiven und deklarativen Formalismus.

Obwohl es im statischen Fall weitreichende Möglichkeiten gibt quantitative Schlussfolgerungen zu ziehen ist dies im Zeitlichen nicht der Fall. Darüber hinaus gibt es (i) viele verschiedene Formen quantitativer Schlussfolgerungen, die (ii) nicht trivial in den zeitlichen Kontext integriert werden könen, da das zum komplexen Zusammenspiel zwischen zeitlichen und quantitativen Schlussfolgerungen kommen kann. Wir untersuchen daher die Kombination zeitlicher und quantitativer Erweiterungen ASPs.

Zunächst führen wir zwei sehr allgemeine quantitative Erweiterungen von ASP ein, indem wir eine *algebraische Semantik* für die quantitativen Aspekte definieren, die auf *Halbringen* basiert. Diese Erweiterungen fügen einerseits quantitative Aggregationsfähigkeiten über die Menge der Modelle und andererseits präzise Spezifikationsmöglichkeiten für quantitativen Bedingungen innerhalb des Programms hinzu. Für Beide analysieren wir ihre Beziehung zu früheren Formalismen mit ähnlichen Möglichkeiten und zeigen, dass Unsere eine konservative Erweiterung ist. Schließlich kombinieren wir die quantitativen Erweiterungen mit einem zeitlichen Ansatz namens LARS, um allgemeines quantitatives Stream Reasoning zu ermöglichen.

Abgesehen von ihrer Definition untersuchen wir die Erweiterungen und den allgemeinen Ansatz im Hinblick auf ihre theoretischen Eigenschaften, einschließlich der Komplexität typischer Schlussfolgerungsaufgaben, der Variablensicherheit von Fragmenten und der Ausdrucksstärke.

Zweitens stellen wir fest, dass beide Erweiterungen das Lösen gewichteter Modellzählungsprobleme über Halbringen beinhalten. Interessanterweise hängt die Komplexität des Problems vom jeweiligen Halbring ab. Diese Abhängigkeit ist jedoch noch nicht charakterisiert worden. Wir liefern eine Charakterisierung anhand einer Familie neuer Komplexitätsklassen und setzen diese in Verbindung zu bekannten klassischen Komplexitätsklassen.

Zu guter Letzt betrachten wir wie man unseren Ansatz in der Praxis anwenden kann. Dabei betrachten wir nur einen begrenztes Fragment, indem wir quantitative Bedingungen und zeitliche Aspekte auslassen und uns auf die Aggregation von Informationen über die Menge der Modelle konzentrieren. Durch eine Mischung aus bekannten Ergebnissen aus der Literatur und neuen Erkenntnissen liefern wir eine Implementierung, die mindestens mit dem Stand der Technik im probabilistischen Schließen mithält und sogar eine bessere Leistung bei zyklischen Instanzen bietet.

Contents

CHAPTER 1

Introduction

We as humans face and routinely solve problems that involve reasoning in a multitude of domains. Consider for example the following question:

> If I leave now and take a given series of metro connections, how likely is it that I will arrive at the store before it closes?

This question concerns a *temporal domain* and involves quantitative reasoning both in the form of *handling the uncertainty* that arises from possibly delayed metros as well as the *numeric calculations* necessary to calculate the time necessary for the trip in dependence on estimated metro departure times.

We can quickly come up with an approximate answer based on experience, intuitions, and rules of thumb and this suffices for the typical questions of minor importance that we are faced with in everyday life. The same cannot be said for many industrial problems - they can be very hard to solve or approximate even given years of experience but mistakes can have detrimental effects. Thus, it is desirable to be able to solve such problems in an automated manner with computers.

Naturally, letting computers solve any kind of problem in any combination of domains is unrealistic. We focus instead on the temporal and quantitative domain, as well as their combination. In order to bridge the gap between how humans think and how computers work, we use *declarative programming* in the form of Answer Set Programming (ASP) [EIK09; Lif08] as the basis of our investigations and consider how it can be extended to provide temporal and quantitative reasoning capabilities.

1.1 Declarative Programming with Answer Set Programming

In order to solve problems using declarative programming, contrary to solving them with imperative programming, one does not need to specify explicitly which steps are performed in order to obtain a solution. Instead one states, using a logical theory, which axioms a solution to a given problem has to fulfill.

A prominent declarative programming method is Answer Set Programming (ASP) [EIK09; Lif08], where the logical theory consists of rules of the form

$$r = a \leftarrow b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m.$$

Such a rule can be intuitively read as "If b_1 to b_n are known to be true and c_1 to c_m are not known to be true, then we know that a is true". Observe that instead of requiring c_1 to c_m to be "known to be false" we only required that we do not know them to be true. This is because ASP uses a Closed World Assumption (CWA) meaning that propositions which are not known to be true are assumed to be false. The negation not c is interpreted as failure to derive c. Therefore, the models, so called *answer sets*, of programs are such that if there is no reason to believe that something is true, then it will not be asserted by the answer set.

The fact that ASP uses a CWA makes it similar to intuitive human reasoning which may derive information like "Edward is not far- or nearsighted" from the fact that "Edward does not wear glasses" and retract it when it becomes clear that "Edward forgot his glasses at home". This could be captured in the program

$$\begin{aligned} \text{impaired\_sight(Edward)} &\leftarrow \text{glasses(Edward)} \\ \text{glasses(Edward)} &\leftarrow \text{wears\_glasses(Edward)} \\ \text{glasses(Edward)} &\leftarrow \text{forgot\_glasses(Edward)} \end{aligned}$$

It may be due to this similarity that declarative programming with answer set semantics has been used in many industrial applications [EGL16; Fal+18].

1.1.1 Extensions of ASP

While propositional ASP is already NP-hard and therefore powerful enough to express many challenging problems, their specification can be somewhat tedious and complicated. Further, some relevant problems have higher complexity than those expressible in ASP or require reasoning over dynamic data, i.e. data that changes with time. The practical usage of ASP gave rise to a need for extensions of the specification language making programs simpler, more expressive and more concise [AF18; Del+03]. Thus, ASP was extended in multiple dimensions.

Figure 1.1 visualizes some of the many ways ASP was extended. The directions we consider and refer to in the figure are the following:

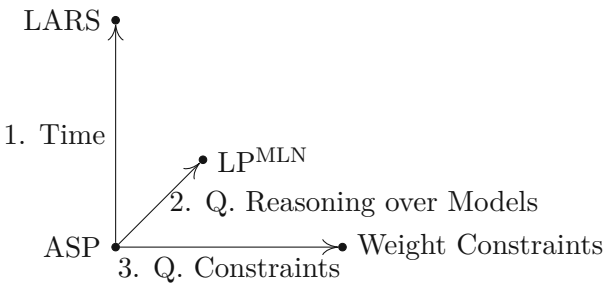


Figure 1.1: (Some of the) directions that ASP was extended along.

1. Time Domain: In [Bec14; BDE18] ASP-semantics were combined with a temporal context resulting in the Logic-based framework for Analytic Reasoning over Streams (LARS). Here, one does not consider a static set of fact but considers interpretations that assign each considered timepoint a possibly different set of facts. Furthermore, operators to capture temporal aspects were added. For example the operator $\Diamond$ corresponds to existential quantification over the time points. Apart from LARS, there are also many other prominent temporal extensions of ASP, such as Temporal Equilibrium Logic (TEL) [Cab+18; CV07; Agu+13; Agu+17], Metric Equilibrium Logic (MEL) [Cab+20a; Cab+22], Temporal Answer Sets (TAS) [GMD13], Temporal Datalog [Ron+18], and DatalogMTL [Wal+19; Wal+21].

2. Quantitative Reasoning over Models: Given an answer set program, we may not only be interested in its answer sets but also some *quantities* associated with them. Extensions of ASP in this direction led, among others, to semantics that enable:

   2.1 Probabilities of Models [BGR09; NM14; LY17; NM15; DKT07];

   2.2 Quantitative Queries [FFP10];

   2.3 Preferences over Models [LY17; BLR97; Leo+06];

   2.4 Weighted Model Counting [KVD11].

3. Quantitative Constraints: Much research has gone into simplifying the specification language of ASP and improving its succinctness. A common strategy is to add more elaborate *constraints* over some quantity as basic language elements. Among others, the following constructs were introduced:

   3.1 Aggregates [Fer11; Del+03; EPS04; Gre99; Leo+06];

   3.2 Weight Constraints [FL05; SNS02; NSS99];

   3.3 Arithmetic Operations [Leo+06; Lie14];

   3.4 Guessing in Rule-Heads [NSS99; Lie14].

This significantly enhanced the expressive power of programs and facilitated their specification. Naturally, we would like to use combinations of these enhancements. For example,

to answer the question of the timely arrival given a series of metro connections, we need to employ probabilistic reasoning (2.1) over temporal data (1.), while adhering to a set of complex constraints over quantities (3.2).

1.1.2 State of the Art

In short, our goal is to enable reasoning that is (1.) temporal and quantitative (2. and 3.) at the same time in a common unified framework with ASP semantics. To the best of our knowledge such a framework does not exist yet. We instead discuss here existing work that is either (a) a highly general extension along direction 2. or 3. that captures many of the features in that direction or (b) a framework that combines quantitative and temporal features.

The state of the art regarding extensions that focus on extending a particular aspect are discussed in the sections they are relevant to (see Section 2.4, Section 2.2.2, and Section 2.3.4 for extension in direction 1., 2., and 3., respectively).

Highly General Quantitative Extensions

ASP with CLP [EPS04] (direction 3.) Elkabani et al. connected ASP with support for generic constraint domains and instantiate this framework with constraints involving aggregates. The framework is mostly practically oriented and includes an implementation with commonly used aggregates which can arbitrarily extended to further aggregates. While the level of generality is quite high, the two variants of the answer set semantics which were introduced are not entirely satisfactory. The first variant ignores constraints completely when checking whether asserted facts are supported by some derivation, the second requires the translation of the program into one without constraints and aggregates, which may end up being exponential in the size of the original program.

Nested Expressions [Fer11] (direction 3.) Ferraris' approach of adding the possibility to have nested constraints on aggregates is quite general, not only with respect to the aggregates that can be used but also with respect to expressing other quantitative extensions in it. There is one restriction on aggregates, namely that they have to be over the real numbers. Three other quantitative extensions have been shown to be partly subsumed by the approach.

Hybrid ASP [Cab+20b; Cab+20c] (direction 3.) Cabalar et al. defined a extension of HT Logic that includes highly general constraints and multi-valued interpretations and allows for handling numerical values in ASP. The approach integrates conditionals and aggregates. The definition of constraints is general enough to fit many specific features that were introduced in previous extensions, such as aggregates, weight or choice constraints. However, it requires extra definitions on the meta level (contrary to only definitions in a predefined language used to specify answer set programs) to enable the usage of such constraints in the specification of programs. While these necessary

definitions are given for constraints over the real numbers they are left open for quantities that are not reals.

LP$^{\text{MLN}}$ [LY17] (direction 2.) Lee and Yang's extension of logic programming with probabilistic reasoning has been shown to subsume two other model level reasoning extensions, namely ASP with weak constraints [BLR97] and P-log [BGR09]. This shows that with LP$^{\text{MLN}}$ one is not only capable of drawing probabilistic inferences in a general way, but also to formulate optimization problems over answer sets as with ASP with weak constraints. It however completely neglects quantitative queries. Furthermore, it is only possible to specify optimization problems where the objective function is mapping to the reals.

Algebraic Prolog [KVD11] (direction 2.) Kimmig et al.'s extension of Prolog defines an algebraic semantics that makes use of semirings. Here, by choosing a suitable semiring we are able to perform probabilistic queries, preferential reasoning, parameter learning and much more. Thus, this extension offers all the capabilities along direction 2. that we desire to have. However, it is defined not for ASP but Prolog. While the semantics of the two declarative reasoning frameworks are arguably similar there are still notable differences.

Semiring-based Constraint Satisfaction Problems (SCSPs) [BMR97] (direction 2.) Similarly to algebraic Prolog, SCSPs have highly general semantics that capture many other previous quantitative extensions of CSPs. Again, this is enabled by using semirings. However, the semantics is for CSPs and not ASP. Additionally, only restricted semirings are allowed such that for example probabilistic inference is not possible with SCSPs. Semiring-based Constraint Logic Programming [BMR01] is related but uses multi-valued interpretations to assign variables a numerical value rather than enabling quantitative reasoning over the set of models.

Combining Frameworks

telingo [Cab+18] Telingo is a solver from the Potassco family [Geb+11], which solves a fragment of TEL. It combines temporal reasoning (direction 1.), quantitative reasoning over the set of models (direction 2.), and quantitative constraints (direction 3.), however it does not yet incorporate the temporal aspects fully in the quantitative constraints. Additionally, its capabilities in direction 2. are limited to preferential reasoning and model enumeration, rather than supporting probabilistic reasoning and other tasks.

General Annotated Logic Programs (GALP) [KS92] GALPs have multi-valued interpretations and allow for the specification of values using arbitrary monotone functions and for limited temporal reasoning. But, while they are also quantitative and temporal they perform a different kind of quantitative reasoning than that of both direction 2. and

3. Furthermore, they are less expressive when it comes to temporal reasoning compared to other temporal extensions of ASP such as LARS or TEL.

Semantic Stream Reasoning (SSR) [PE20; PET21] SSR combines the temporal domain and quantitative reasoning over the set of models, specifically to enable parameter learning and prediction tasks over temporal streams. While the semantics are not explicitly discussed it also allows for quantitative constraints in the program. This work focuses on solving a specific type of problem by combining capabilities of the directions 1., 2., and 3. rather than introducing a general framework for solving different kinds of reasoning tasks that need features from all the different directions.

Probabilistic ASP (PrASP) [NM14] PrASP allows for probabilistic reasoning (direction 2.) in the context of web stream reasoning (direction 1.). However, its semantics do not consider the handling of quantitative constraints in the temporal context. Additionally, the focus here is only on probabilistic inference rather than general quantitative reasoning over the set of models.

Summary

There are other extensions with similarly strong capabilities, however, the main strengths and shortcomings are similar to the ones of the extensions presented above. Overall, we see that there are some quite general extensions, which are able to subsume many others in them. Even further, there are even some frameworks that support reasoning that combines the capabilities of the three direction. Nevertheless, the combined capabilities are all limited in some way or the other. The proper strategy of completely incorporating the temporal domain into quantitative reasoning over the set of models and quantitative reasoning is not yet clear.

1.2 Problem Statement

We are interested in a general framework which combines 1., 2., and 3., that is, we want to

Find and analyze a general framework that allows for succinct specifications and reasoning over answer sets in a streaming context.

When *finding* such a general framework we are faced with the following two main challenges. On the one hand, we have seen that Figure 1.1 oversimplifies the different directions: there are many extensions along 2. and 3. that actually tackle different sub-problems. They still go in the same general direction but the specific purposes do not coincide exactly. We adapt the approximate visualization of Figure 1.1 to reflect this in Figure 1.2.

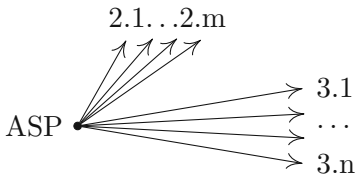


Figure 1.2: Multiple extensions 2.i and 3.j along the directions 2. and 3.

This entails that we need to find a reasonable way to capture all the capabilities introduced along 2. and 3. Reasonable here meaning a strategy different from simply taking an unstructured "union" of the extensions 2.1 to 2.m and 3.1 to 3.n, respectively, and different from a restriction to a significantly less powerful subset of features by choosing 2.i, 3.j and extending them to the stream reasoning context.

On the other hand, we need to incorporate the temporal domain. For this purpose one can not reuse previous definitions in a straight forward way. Consider for example constraints on aggregates in a temporal context. Here, one needs to be able to specify whether the aggregation should consider only the values at the current time point or at any time point. Further, one needs to be able to specify how duplicate occurrences of a value at different time points should be handled. Similar problems occur also with other capabilities along the directions 2. and 3.

In summary, we need to find frameworks $?_{2.}$ and $?_{3.}$ that can be combined with a suitable temporal extension of ASP, such as LARS, resulting in $?_{1.+2.+3.}$, as is visualized in Figure 1.3.

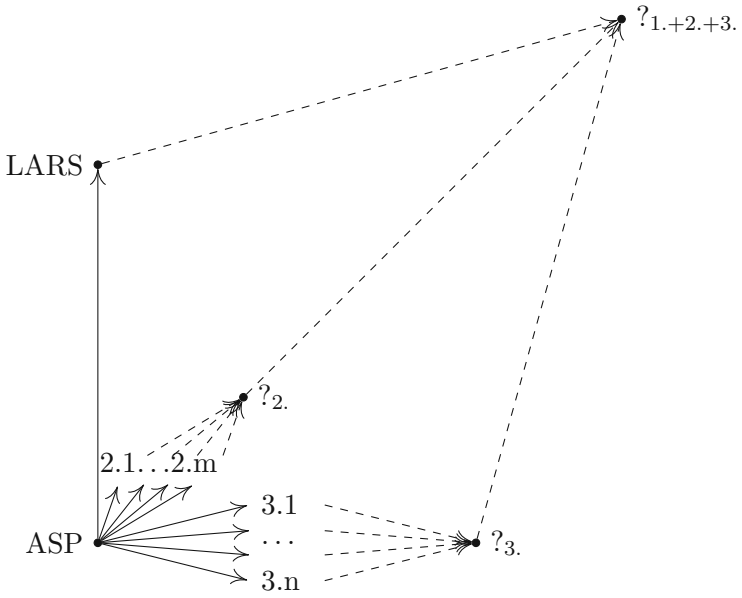


Figure 1.3: "Questionmarks" that need to be filled in.

Given that we find appropriate answers for the questionmarks, we further need to *analyze* the resulting framework with respect to different aspects. This is necessary in order to make the framework practically useful. An implementation and its application to real world use cases require an analysis of efficiently solvable and safe fragments of the general framework.

Detailed research questions that need be answered in the process of finding and analyzing such a framework are given after a discussion of the methodology we use.

1.2.1 Approach

Both the extensions along 2. and the ones along 3. are *quantitative*, i.e. involve some form of calculations.

A semiring $\mathcal{R}$ is an algebraic structure of the form $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ where one has some form of addition $\oplus$ and multiplication $\otimes$ with neutral elements $e_\oplus$ and $e_\otimes$ respectively. Using semirings one can capture various modes of computation ranging from the Boolean semiring $\mathbb{B} = (\{\bot, \top\}, \vee, \wedge, \bot, \top)$ over the min-plus semiring $\mathcal{R}_{\min,+} = ([0, \infty], \min, +, \infty, 0)$ to the well known semiring over the real numbers $\mathbb{R} = (\mathbb{R}, +, \cdot, 0, 1)$. In their seminal paper [DG07], Droste and Gastin found a way to connect calculations in semirings and logical formulas, introducing *weighted logics*. In weighted logics, formulas over some semiring $\mathcal{R}$ can contain elements of the semiring as subformulas in addition to their normal syntax. Instead of Boolean semantics, the semantics of weighted formulas is their interpretation as algebraic expressions over a semiring. This is achieved by interpreting disjunctive connectives as addition $\oplus$ and conjunctive connectives as multiplication $\otimes$. In line with this, atomic formulas which are false with respect to a given interpretation are interpreted as $e_\oplus$ corresponding to the empty disjunction or sum and true atomic formulas are interpreted as $e_\otimes$. Negation is similar to the complement of the truth value in the sense that the negation of $e_\oplus$ is $e_\otimes$ and negation of any other value is $e_\oplus$. Consider for example the weighted formula

$$\alpha = 15 * \text{Circus} + 20 * \text{Restaurant}$$

over the semiring $\mathbb{R}$ encoding the amount of money spent on an evening depending on which places are visited. With respect to an interpretation satisfying Restaurant but not Circus the formula has semantics

$$15 \cdot 0 + 20 \cdot 1 = 20.$$

If we, however, interpret α as a formula over the min-plus semiring $\mathcal{R}_{\min,+}$ encoding that the cheapest prize possible when one wants to plan an evening, depending on which activities are possible, we obtain under the same interpretation

$$\min(15 + \infty, 20 + 0) = 20.$$

Semirings have already previously been successfully applied in situations where one may need to perform calculations using different operations and objects but of similar

structure. Examples range from parsing [Goo99] to weighted model counting [KVD11] and weighted abstract argumentation [BRS18]. In another work, Green, Karvounarakis, and Tannen [GKT07] used them to define provenance semantics for database queries. Here, Green et al. showed that different kinds of quantitative settings can all be expressed using relational algebra queries with provenance semirings. Namely, they showed that bag semantics, probabilistic databases and why-provenance as well as the classic relational algebra queries can all be defined using the same semantics over different semirings.

We aim to use the same idea to capture directions 2. and 3. For direction 2., the quantitative reasoning over models, it is sufficient to use the classical weighted semantics of formulas over semirings. When it comes to 3., the quantitative constraints, it is necessary to define answer set semantics that are in line with the closed world assumption of ASP also for weighted formulas. This is because for quantitative constraints the weighted formulas are part of the program and influence which atomic formulas are asserted in answer sets.

There are many common approaches for the definition of answer set semantics. In general, these approaches have to ensure that the CWA used in ASP is captured correctly, meaning that one should only satisfy atomic formulas in an answer set if one is forced to do so by the program. The first approach we consider uses the notion of *reducts* [Lif08]. There are different kinds of reduct-based approaches; we focus on the Gelfond-Lifschitz reduct. Here one defines that for programs without negation the (unique) answer set $Cl(\Pi)$ is the interpretation that satisfies the program Π and is the subset minimal one among the interpretations doing so. An interpretation $\mathcal{I}$ that satisfies a program Π with negation is then an answer set of Π if it is the answer set of $Cl(\Pi^{\mathcal{I}})$. Here $\Pi^{\mathcal{I}}$ is the reduct of Π with respect to $\mathcal{I}$ where all negated formulas are replaced by their truth value with respect to $\mathcal{I}$. This results in a program without negation, where the operation Cl is again defined.

We also consider the answer set semantics defined using Here-and-There (HT) Logic [Pea06]. In this approach, which can be seen as a fragment of intuitionistic logic or the multi-valued logic N_5 [PV04], one considers two worlds H and T having one interpretation each, viz. - *here* $\mathcal{I}^H$ and *there* $\mathcal{I}^T$, respectively, where $\mathcal{I}^H$ is a subset of $\mathcal{I}^T$. The truth value of a formula with respect to $(\mathcal{I}^H, \mathcal{I}^T)$ is then defined in such a way that negated formulas are assigned the truth value that they have with respect to $\mathcal{T}$. This leads to a semantics that is equivalent to the reduct-based semantics on classic answer set programs.

HT Logic has the benefit that its definition is rather generic and can thus be generalized to logics that are different from propositional logic easily. In fact, multiple extensions of ASP are based on HT Logic [Cab+22; Cab+18; ES22; FL10; Lif21; LTT99]. Thus, it is a more promising starting point for the definition of an answer set semantics of weighted formulas.

1.2.2 Research Questions

Weighted logic combines Boolean logic and algebraic expressions leading to the possibility to specify calculations dependent on the satisfaction of logical formulas with ease. In

Weighted HT Logic, this dependency can be adapted to follow the intuition of answer set semantics. On the one hand, Weighted Logic allows many different modes of calculation due to the variability of the underlying semiring. On the other hand, its syntax and semantics is homogeneous for each of those modes. This makes Weighted (HT) Logic a promising approach to formalize many quantitative extensions of ASP along both 2. and 3. using a common homogeneous language which is still simple enough both for practical applications (i.e. actual usage) and theoretical analysis (i.e. the study of properties and algorithms).

Furthermore, Weighted Logic is rather generic in the sense that for many Boolean logics it is possible to define a weighted version, by using the strategy described in Section 1.2.1 (i.e. conjunction is multiplication, disjunction is addition). This is beneficial when it comes to the combination with the temporal domain.

We aim at a solution of the problem stated in Section 1.2 using Weighted (HT) Logic. Thus, we consider the following research questions, visualized in Figure 1.4:

(i) Generalization & Faithfulness

(Q1) Can we use Weighted Logic for General Quantitative Reasoning over Models?

(Q2) Can we use Weighted HT Logic for General Quantitative Constraints?

(Q3) Can we capture temporal aspects using Weighted (HT) Logic?

The questions under (i) ask whether we can use Weighted (HT) Logic to successfully fill in the questionmarks in Figure 1.3. Q1 asks whether ASP in combination with Weighted Logic can take the place of $?_{2.}$. Q2 asks whether ASP in combination with Weighted HT Logic can take the place of $?_{3.}$. Q3 asks whether lifting the combination of ASP and Weighted (HT) Logic to a combination of a temporal logic such as LARS or TEL and Weighted (HT) Logic is sufficient to capture the additional temporal aspects that are relevant in the streaming domain.

Successfully means two things here. The first is *Generalization*, i.e., the capabilities of previous extensions are subsumed (to a reasonable degree). The second is *Faithfulness*, i.e., the semantics is a conservative extension of the semantics of previous works (to a reasonable degree), such that features of previous extensions have the meaning that is expected of them.

If questions Q1 to Q3 are answered positively, we have completed the finding part of our problem in Section 1.2. What is left is a thorough analysis.

(ii) Theoretical Analysis

(Q4) What does the complexity of reasoning depend on?

(Q5) Which properties of programs carry over and which new ones can be found?

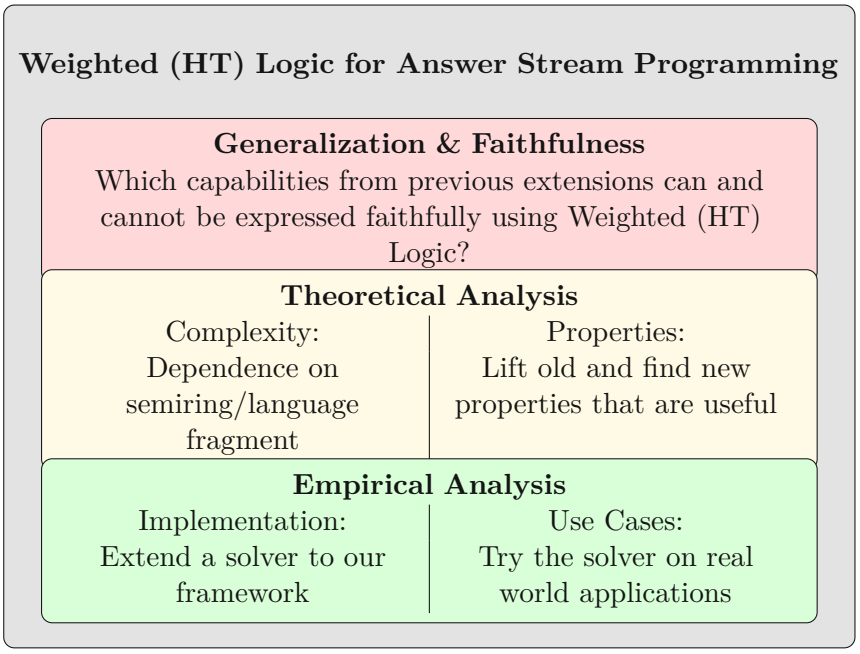


Figure 1.4: The research questions and their division into subquestions.

(iii) Empirical Analysis

(Q6) To what extent can we efficiently implement our framework?

(Q7) How can our framework be employed in real world applications?

1.3 Contributions and Thesis Structure

The contributions in this thesis and the structure of it closely follow the research questions posed in Section 1.2.2. Therefore, we discuss them together in this subsection. Furthermore, we indicate which parts of the contents of this thesis are based on which publications.

Generalization & Faithfulness. The most important aspect of using Weighted (HT) Logic for quantitative capabilities is the question of generalization and faithfulness. If we were unable to express the desired capabilities faithfully, a further consideration would be pointless and we would need to start from scratch. Therefore, this topic is discussed first.

These aspects are considered in Chapter 2 of this thesis, which is on "General Quantitative Stream Reasoning". The contents of this chapter are mostly based on previously published conference papers. That is, Section 2.2 on "Model Level Quantitative Reasoning" is based on [EK20b], Section 2.3 on "Truth Level Quantitative Reasoning" is based on [EK20a] and only Section 2.4 on "Combining Stream Reasoning and Quantitative Reasoning" is

mostly novel and but draws inspiration from [EK20b] for the definition of weighted LARS formulas and the complexity and expressivity results.

In Chapter 2, we first discuss preliminaries such as ASP, Semirings and Weighted Logic. Based on this, in Section 2.2, we introduce a general framework for quantitative model level reasoning (direction 2.) called *algebraic measures*. Intuitively, algebraic measures combine a logical specification of a background theory, in our case an answer set program, that specifies which interpretations are models with a quantitative specification, in the form of a weighted formula, which assigns each model a weight over a semiring. As the following comparison with related formalisms shows, we can use these measures for probabilistic reasoning, preferential reasoning, quantitative queries and weighted model counting by employing an appropriate semiring. Namely, this is achieved by respectively normalizing, optimizing, simply used to retrieving a quantity or summing up their value over all answer sets of a program. Most importantly, we see that algebraic measures are a suitable solution to capture extensions along direction 2., since they are both highly general as well as uniform due to the parameterization of their definition by semirings.

Next, in Section 2.3, we introduce a general framework for quantitative constraints (direction 3.) called *algebraic constraints*. Here, we switch from ground ASP to programs with variables, since these are typical and often intertwined with the specification of quantitative constraints or aggregates. Since we want to make use of Here-and-There Logic, we introduce their semantics via the HT semantics of first-order formulas. Similarly to algebraic measures, algebraic constraints are also based on Weighted Logic, however, like our programs, they are not propositional but first-order and have HT semantics. The resulting first-order Weighted Here-and-There formulas allow us to specify quantitative computations with variables elegantly and in *non-monotonic* dependency on the truth of atomic formulas. By allowing constraints of the form $\alpha \sim_{\mathcal{R}} k$ for a weighted formula α, relation $\sim$, semiring $\mathcal{R}$ and semiring value k the extension to answer set programs with algebraic constraints (ASP($\mathcal{AC}$)) can express quantitative constraints over different domains and semirings with ease. Indeed, the definitions are followed by an analysis of the expressivity of ASP($\mathcal{AC}$) compared to other extensions of ASP with quantitative constraints. This analysis reveals that and how well-known constructs, such as aggregates, weight constraints, different forms of conditionals and choice constraints, are subsumed by algebraic constraints. Apart from the subsumption of previous construct we also find that so called *minimized constraints* are a novel construct that can be expressed with algebraic constraints.

A practical example confirming the benefit of ASP($\mathcal{AC}$) is its capability to model provenance queries, which is demonstrated in Section 2.3.2. In order to allow for an informed usage of ASP($\mathcal{AC}$), we consider some aspects of its language in Section 2.3.3 such as safety and strong equivalence. Notably, safe programs are domain independent and thus guarantee intuitive behavior that does not depend on the set of constants we assume to exist. Last but not least, we provide an analysis of the computational complexity that comes with algebraic constraints in Section 2.3.5. This analysis shows that when the considered program is essentially propositional (i.e. ground), the complexity of disjunctive

logic programs is retained and model checking (MC) and strong equivalence (SE) are coNP-complete while answer set existence (SAT) is Σ_2^p-complete, provided the semirings that are used satisfy a practically mild encoding condition. For safe non-ground programs, MC is feasible in EXPTIME at most, while SAT and SE are undecidable in general. A more in depth consideration however reveals that there are expressive decidable fragments. The latter two subsections go beyond the scope of generalization and faithfulness and instead are part of the theoretical analysis.

Sections 2.2 and 2.3 answer the questions (Q1) and (Q2) regarding generalization and faithfulness positively. Accordingly, Section 2.4 combines algebraic measures and constraints with the temporal reasoning capabilities in the form of the LARS framework resulting in Algebraic LARS. For this, we first recall the standard semantics for LARS as well as its HT-semantics in Section 2.4.1. Notably, we define the HT-semantics by introducing a temporal extension of the first-order sorted Here-and-There Logic that we used for the definition of ASP($\mathcal{AC}$) and embedding LARS formulas into such *Timed* first-order sorted Here-and-There formulas. This makes the lifting of algebraic constraints to LARS, which follows in Section 2.4.2, straightforward. Here, we also give a final version of the running example regarding metro connections, which highlights the strengths of the combination of algebraic constraints, algebraic measures and LARS. Since the complexity of Algebraic LARS is strictly higher than that of ASP($\mathcal{AC}$), which is already undecidable in the non-ground case, we only consider the expressivity and complexity of LARS measures over ground programs without algebraic constraints. For the expressivity, we show equivalence to Weighted Automata and weighted Monadic Second-Order Logic in Section 2.4.3. For the complexity in Section 2.4.4, we show FPSPACE-completeness for the evaluation problem over a specific semiring and FPSPACE-membership under restrictions on the semiring. Furthermore, we show a range of completeness results for preferential reasoning with LARS measures. Finally, we conclude this section in Section 2.4.5.

A visual overview of the results obtained in this thesis towards a general framework for quantitative stream reasoning under answer set semantics is given in Figure 1.5.

We direct our attention to the follow up questions.

Theoretical Analysis. Given that we have established a faithful and general framework for the combination of quantitative and temporal reasoning aspects, we are of course interested in its properties in the next step. For ASP, the following properties are known to be of interest from previous work.

- **Safety**, a property necessary for ASP with variables that ensures that programs do not have unintended consequences but only those envisioned by the "programmer". Safety was considered, inter alia, in [Geb+15; Lif16; CPV09; Eit+13b; Cab11; LLP08; Eit+08].
- **Finite Groundability**, a stronger restriction than safety that entails that there is a finite domain that can be used to evaluate programs. Without this restriction

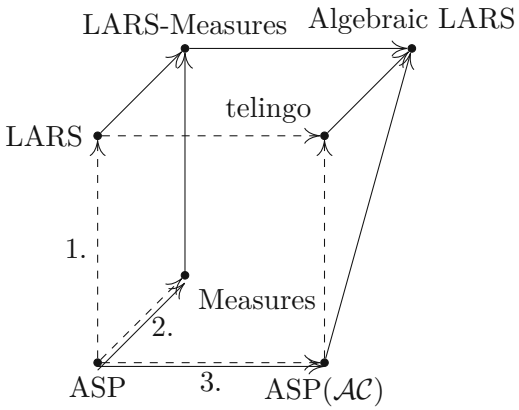


Figure 1.5: Overall structure of the generalization and faithfulness results. Vertices correspond to frameworks that have the capabilities added by each edge pointing to them. Dashed edges indicate previous work of other authors that we know of, solid arrows correspond to a contribution in our work (novel or improved extension in this direction).

practical reasoning problems are often undecidable. Even if they are decidable, matters are complicated since one needs to draw conclusions about programs with infinitely many rule. Finite groundability was considered, inter alia, in [LL09; Eit+13a; BL10; Red17; AZZ17; Eit+16; Cal+08b].

- **Program Equivalence**, a property that guarantees that programs can be replaced with one another without changing their consequences. It is used for example in the simplification of answer set programs. Program equivalence was considered, inter alia, in [NL15; LPV01; BDE16; CNR22; Lie22; Wan+21; FMW19; CD14; Fin11].

- **Computational Complexity**, i.e., a characterization of the theoretical hardness of a computational problem, such as checking whether a program has an answer set, finding out whether two programs are equivalent or in our context computing the value of an algebraic measure. Knowing the computational complexity is useful, since it provides a way of judging how hard it is to solve a problem and whether a given algorithm is suitable for the solution of a problem. Furthermore, its analysis is often based on problem reductions and, thus, can be used as a basis for algorithms that reduce the problem to another for which an efficient algorithm is known or an implementation exists. Computational complexity was considered, inter alia, in [Eit+04; Dan+01; CM20; AL15; Blo+14; AF13; FPL11; Eit+07].

Especially the computational complexity that comes from working with semirings is of interest for us. Therefore, we discuss the other properties in close vicinity of the introduction of our extension(s) of ASP but dedicate a separate chapter to the complexity of counting over semirings.

Therefore, these aspects are considered partially in Chapter 2, which is on "General Quantitative Stream Reasoning" and partially in Chapter 3, which is on "Complexity of Counting over Semirings". As discussed above, the content of Chapter 2 is mostly based on previously published conference papers. That is, Section 2.2 is based on [EK20b], Section 2.3 is based on [EK20a] and only Section 2.4 is novel although it draws inspiration from [EK20b]. The contents of Chapter 3 are mostly based on [EK21] and the journal article [EK23].

Safety conditions for ASP($\mathcal{AC}$) as well as the strong equivalence of ASP($\mathcal{AC}$) programs are considered in Section 2.3.3. Furthermore, for finite groundability of ASP($\mathcal{AC}$) programs, we also provide an appropriate definition by combining the concept of domain restrictedness from [NSS99] and argument restrictedness from [LL09], resulting in a class of finitely ground programs, for which deciding membership is tractable. This can be found in Section 2.3.5. There, we also discuss the computational complexity of different reasoning tasks for ASP($\mathcal{AC}$) that are not related to counting over semirings.

For the computational complexity associated with counting over semirings, as it happens when we evaluate algebraic measures, we do not restrict ourselves to the counting problems we introduced but also consider a wide range of other problems [KVD11; KVD17; SI96; GKT07; BMR97] from the literature that are also defined over semirings. Here, the complexity is also unknown but of general interest to the scientific community, which is why we proceed with an in depth complexity analysis as follows.

First, we go over the necessary preliminaries of propositional logic, semirings and their properties, and classical complexity classes, which we use to characterize the hardness of the counting problem over semirings in Section 3.2. After this, we provide a toolbox for the complexity analysis of counting problems over semirings. For this, we generalize the well-known complexity class NP to NP($\mathcal{R}$), which is an analogue of NP and #P over a semiring $\mathcal{R}$. We complement this by an NP($\mathcal{R}$)-complete problem that we name SAT($\mathcal{R}$) defined as the evaluation problem of weighted existentially quantified Boolean formulas. A final discussion of alternative definitions of NP($\mathcal{R}$) concludes the introductory phase in Section 3.4. We immediately put the established toolbox to use in Section 3.5 to show that a wide range of problems, including overall weight queries for algebraic measures and the evaluation of the weighted formulas in algebraic constraints, are NP($\mathcal{R}$)-complete. Interestingly, we also found one problem over semirings, namely provenance queries [GKT07], that is NP($\mathcal{R}$)-hard but not complete in the general setting. This provides the first complete characterization of the computational complexity of the discussed problems. Since NP($\mathcal{R}$) is a new complexity class, it may give some insight in the type of computations that the aforementioned problems require but does not give much intuition for the actual algorithmic hardness. We therefore relate NP($\mathcal{R}$) to well-known complexity classes from the literature in Section 3.6. An immediate problem here is that semiring values need to be encoded in a finite alphabet and calculations with them need to be performed in the alphabet representation. This is inconvenient, since the complexity can be easily shown to depend on the encoding. We address this issue by only considering *efficiently encoded* semirings. Our following analysis reveals that NP($\mathcal{R}$)

is at least as powerful as NP or Mod_pP regardless of the semiring, for some semirings we can even show that they require an exponential size output unless the polynomial hierarchy collapses. Such a collapse is widely assumed to be unlikely. We can, however, also give upper-bounds on the complexity showing that for many semirings access to an NP-,Mod_pP- or #P-oracle suffices. Finally, we discuss related formalisms before concluding and discussing future work in this direction.

Overall Chapter 3 provides a detailed account of the computational complexity of counting problems over semirings, which is not only relevant for us but many other semiring frameworks as well. Apart from this it also provides an approach that may prove useful for exploiting semiring properties to design more efficient algorithms.

Empirical Analysis. In order to test the practical applicability of our approach, we also perform an empirical evaluation. Naturally, this necessitates an implementation of our approach. However, a complete implementation of the combination of algebraic constraints, algebraic measures and the temporal aspect would be rather challenging, and would go beyond the scope of this thesis. Since current implementations of ASP with constraints and the possibility of covering temporal aspects at least partially are already rather advanced [Cab+19], we focused on algebraic measures. While an implementation in the probabilistic logic programming solver ProbLog [KVD11; Fie+15] exists, this implementation does not cover general normal answer set programs.

These aspects are considered in Chapter 4, which is on "Efficient Algebraic Answer Set Counting". The content of Chapter 4 is mostly based on [EHK21] and the manuscript [EHK23].

Generally, in Chapter 4, we first discuss which aspects need to be taken into account in an implementation. We then describe the implementation and finally provide an in depth experimental evaluation, where we compare different configurations of our implementation and other solvers that constitute the current state of the art in the area.

We first discuss the necessary preliminaries about algebraic measures, semirings, propositional logic and knowledge compilation in Section 4.1. Next, we provide additional motivation for an implementation by discussing applications of algebraic measures in Section 4.3. Then, we discuss the state of the art and derive the possible performance guarantees that current implementations could give and practically give in Section 4.4. This allows us to make an informed decision regarding the strategy that we want to use in our implementation. This investigation reveals that the most promising approach is to use state of the art knowledge compilers [Dar04; OD15] by first translating programs into Conjunctive Normal Form (CNF) formulas. One parameter that can lead to good knowledge compilation performance, apart from small problem size, is the treewidth of the formula [Dar04]. We therefore consider a way to translate any normal answer set program into a CNF, while limiting the increase of the treewidth and the size of the CNF compared to the program. This is discussed in Sections 4.5 and 4.6. Concretely, it requires handling cyclic dependencies in Section 4.6 and a treewidth-aware version of Clark's Completion [Fag94] in Section 4.5.

Next we explain how we get from these theoretical advances to a practically usable implementation in Section 4.7 that we call aspmc, before we thoroughly evaluate aspmc in Section 4.8. Here, we first consider aspmc's different settings to find the best performing approach. Secondly, we compare aspmc's performance using the optimized settings to other implementations that can be seen to evaluate algebraic measures. Last but not least, we discuss our findings in Section 4.9 and mention some possible opportunities for extending aspmc.

Conclusion Finally, in Chapter 5, we revisit the findings of the previous chapters and put them into perspective. That is, we briefly summarize their main results and compare what we aimed to achieve in this work with what we managed to. We end by pointing out some open issues that would be interesting to consider in the future.

Proofs Short proofs are given in full after the statement of the result, long and complicated proofs are only sketched but given in full in Appendix A, Appendix B, and Appendix C for Chapter 2, Chapter 3, and Chapter 4, respectively.

CHAPTER 2

General Quantitative Stream Reasoning

The general goal of this chapter is to find a general framework capable of quantitative reasoning over the set of models, succinct quantitative specifications of the set of models, and reasoning over a temporal domain. Notably, this framework should have a semantics that is in line with the semantics of ASP. Furthermore, we will show that the semantics of this framework is faithful to the semantics of other extensions of ASP, and analyze its properties.

For this purpose, we start out by introducing the basic preliminaries necessary for all the following sections in Section 2.1. This is followed by a general extension of ASP with reasoning capabilities over the set of models in Section 2.2 and a general extension of ASP with succinct quantitative constraint specifications in Section 2.3. Finally, in Section 2.4, we combine the previous two extensions and LARS into a temporal reasoning framework to obtain Algebraic LARS, a framework that satisfies our initial goal. All the last three sections include not only the definition of the extensions but also examples and an analysis of their properties such as safety, expressivity, faithfulness and complexity.

2.1 Preliminaries

At the basis of most of our considerations is Answer Set Programming (ASP). Formally, answer set programs are defined as follows.

Definition 1 (Rule, Answer Set Program)**.** *A normal* answer set program Π *over a set* $\mathcal{P}$ *of predicates,* $\mathcal{V}$ *of variables and* $\mathcal{D}$ *of domain elements is a finite set of* rules r *of the form*

$$a \leftarrow b_1, \ldots, b_m, not\, c_1, \ldots, not\, c_n,$$

where a and all b_j and c_k are atoms *of the form $p(t_1, \ldots, t_l)$ with $p \in \mathcal{P}$, $t_i \in \mathcal{V} \cup \mathcal{D}$, and $l = arity(p)$ the* arity *of the predicate. Given such a rule r, we call $H(r) = a$, $B^+(r) = \{b_1, \ldots, b_n\}$, $B^-(r) = \{c_1, \ldots, c_n\}$ and $B(r) = B^+(r) \cup B^-(r)$ the* head, negative body, positive body *and* body *of r, respectively.*

Intuitively, such a rule r states that for each substitution of the variables in r by elements of $\mathcal{D}$ the head follows from the body.

There are other answer set programs besides normal ones; however, for now, we do not need them. Instead, we introduce them when they become relevant. Nevertheless, we refer to normal answer set programs as answer set programs or simply as programs.

We slightly abuse notation and express constraints by using

$$\leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$$

for

$$\bot \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n, \text{not } \bot,$$

where $\bot$ is a nullary predicate that does not occur in programs otherwise.

Also, we allow *choice constraints*

$$\{p(t_1, \ldots, t_l)\} \leftarrow B^+(r), B^-(r)$$

as a shorthand for the two rules

$$p(t_1, \ldots, t_l) \leftarrow B^+(r), B^-(r), \text{not } np(t_1, \ldots, t_l)$$

and

$$np(t_1, \ldots, t_l) \leftarrow B^+(r), B^-(r), \text{not } p(t_1, \ldots, t_l),$$

for an otherwise unused predicate np.

An example of an answer set program is the following.

Example 1 (Metro Connections)**.** *We consider the variations of following program Π_m throughout this chapter.*

$$reach(S) \leftarrow start(S) \tag{2.1}$$
$$reach(S) \leftarrow exit(M, S) \tag{2.2}$$
$$\{enter(M, S)\} \leftarrow reach(S), depart(M, S) \tag{2.3}$$
$$on\_metro(M, S) \leftarrow enter(M, S) \tag{2.4}$$
$$on\_metro(M, S1) \leftarrow on\_metro(M, S2), next\_stop(M, S2, S1), not\ exit(M, S2) \tag{2.5}$$
$$\{exit(M, S)\} \leftarrow on\_metro(M, S) \tag{2.6}$$
$$\leftarrow enter(M, S), exit(M, S) \tag{2.7}$$
$$\leftarrow enter(M1, S), enter(M2, S), M1 \neq M2 \tag{2.8}$$

$$\leftarrow goal(S), not\, reach(S) \tag{2.9}$$

This program models in which way it is possible to get from a starting station ($start(S)$) to a goal station ($goal(S)$), by using metros that depart at certain station ($depart(M, S)$) and visit one stop after the other ($next\_stop(M, S1, S2)$). Note that in order for the program to be applied the extension of the aforementioned predicates ($start$, $goal$, $depart$ and $next\_stop$) need to be added as input to the program.

Given a reasonable input, Π_m then works as follows: the rules in Equations (2.1) *and* (2.2) *tell us that we can reach a station by starting there or by exiting a metro there. To exit a metro we need to enter it, which Equation* (2.3) *tells us is possible if we reach a station at which the metro departs. When we enter a metro, we keep on being on the metro until we exit it as the rules in Equations* (2.4) *to* (2.6) *specify. Finally, we want to ensure that we do not exit and enter metros unnecessarily, using Equation* (2.7)*, and prohibit multiple metro rides at the same time, using Equation* (2.8)*. The expression $M1 \neq M2$ in Equation* (2.8) *is syntactic sugar for another extensional predicate that is true iff $M1$ is not equal to $M2$. Last but not least, we want to reach our goal and, thus, add Equation* (2.9)*.*

Answer set programs can be seen as a logical specification of a set of solutions. To make the intuition of their meaning clear, we define their semantics, specifying their solutions as so called answer sets:

Definition 2 (Interpretation, Answer Set, Reduct)**.** *We denote by $\mathbf{HB}(\mathcal{P}, \mathcal{D})$ the* Herbrand base *defined over a set $\mathcal{P}$ of predicates and a set $\mathcal{D}$ of domain elements.*

$$\mathbf{HB}(\mathcal{P}, \mathcal{D}) = \{p(d_1, \ldots, d_l) \mid l \in \mathbb{N}_0, p \in \mathcal{P}, d_i \in \mathcal{D}, l = arity(p)\},$$

i.e., the set of all possible combinations of predicates p and with $arity(p)$ arguments d_i from the domain, where $arity(p)$ denotes the arity of the predicate p.

For a program Π defined over $\mathcal{P}, \mathcal{V}$ and $\mathcal{D}$, we use $\mathbf{HB}(\Pi)$ as a shorthand for $\mathbf{HB}(\mathcal{P}, \mathcal{D})$.

The semantics of programs is defined via grounding*. The grounding $ground(\Pi)$ of a program Π over $\mathcal{P}, \mathcal{V}$ and $\mathcal{D}$ is given by the set*

$$ground(\Pi) = \{r\sigma \mid r \in \Pi, \sigma : \mathcal{V} \to \mathcal{D}\},$$

where σ is called a substitution and $r\sigma$ denotes the result of applying σ to r, which means that every variable V that occurs in r is replaced by $\sigma(V)$.

An interpretation $\mathcal{I} \subseteq \mathbf{HB}(\Pi)$ satisfies Π*, if for each rule $r \in ground(\Pi)$ it holds that $H(r) \in \mathcal{I}$ or there exists $a \in \mathbf{HB}(\Pi)$ s.t. $a \in B^+(r) \setminus \mathcal{I}$ or $a \in B^-(r) \cap \mathcal{I}$.*

Furthermore, $\mathcal{I}$ is an answer set *of Π if it is a $\subseteq$-minimal satisfying interpretation of the* reduct *$ground(\Pi)^{\mathcal{I}}$, which is given by*

$$ground(\Pi)^{\mathcal{I}} = \{r \in ground(\Pi) \mid B^+(r) \subseteq \mathcal{I}, B^-(r) \cap \mathcal{I} = \emptyset\}.$$

We denote the set of answer sets of a program Π by $\mathcal{AS}(\Pi)$.

We restrict ourselves to finite domains $\mathcal{D}$ in this section. Sometimes, we refer to answer sets also as stable models.

We consider the semantics again on our example.

Example 2 (cont.)**.** *Before we can ground the program and consider its answer sets, we need to fix a domain and provide some input facts. For the input facts* Π_{in}*, we use a simplified version of the Viennese metro system:*

$$\begin{aligned} start(bsgasse) &\leftarrow \\ goal(tsgasse) &\leftarrow \\ depart(\text{“}U4\text{”}, bsgasse) &\leftarrow \\ depart(\text{“}U1\text{”}, kplatz) &\leftarrow \\ next\_stop(\text{“}U4\text{”}, bsgasse, kplatz) &\leftarrow \\ next\_stop(\text{“}U1\text{”}, kplatz, tsgasse) &\leftarrow \end{aligned}$$

Here, $bsgasse, tsgasse$*, and* $kplatz$ *are shorthands for the stations Braunschweiggasse, Taubstummengasse, and Karlsplatz, respectively.*

Accordingly, it suffices for us to use the domain

$$\mathcal{D} = \{bsgasse, tsgasse, kplatz, \text{“}U1\text{”}, \text{“}U4\text{”}\}.$$

When grounding, we get for the rule

$$reach(S) \leftarrow exit(M, S)$$

among others, the rules

$$\begin{aligned} reach(bsgasse) &\leftarrow exit(\text{“}U4\text{”}, bsgasse) \\ reach(tsgasse) &\leftarrow exit(\text{“}U1\text{”}, tsgasse) \\ reach(kplatz) &\leftarrow exit(\text{“}U4\text{”}, kplatz) \end{aligned}$$

Note that the complete grounding $ground(\Pi_m \cup \Pi_{in})$ *contains more instantiations of this rule such as*

$$reach(\text{“}U1\text{”}) \leftarrow exit(\text{“}U1\text{”}, \text{“}U1\text{”}).$$

However, these rules are always trivially satisfied, since $exit(\text{“}U1\text{”}, \text{“}U1\text{”})$ *is not satisfied in any answer set, as can be easily seen. Thus, we do not need to consider them in the grounding. For space reasons, we do not consider the whole grounding, even without rules that are trivially satisfied.*

Instead, we consider the answer sets of $\Pi_m \cup \Pi_{in}$*. Intuitively, given the input facts we have, there is only one option to get from start to goal, in which we first enter the U4 at bsgasse, then switch to the U1 at kplatzand finally arrive at our goal by exiting at tsgasse.*

We see that the program gives us exactly this unique solution as the answer set $\mathcal{I}_{initial}$ *containing*

$$\begin{aligned}
&enter(\text{“}U4\text{”}, bsgasse), exit(\text{“}U4\text{”}, kplatz),\\
&enter(\text{“}U1\text{”}, kplatz), exit(\text{“}U1\text{”}, tsgasse),\\
&on_metro(\text{“}U4\text{”}, bsgasse), on_metro(\text{“}U4\text{”}, kplatz),\\
&on_metro(\text{“}U1\text{”}, kplatz), on_metro(\text{“}U1\text{”}, tsgasse),\\
&reach(bsgasse), reach(kplatz),\\
&reach(tsgasse),
\end{aligned}$$

in addition to the input facts.

Apart from answer set programs for the logical/qualitative reasoning, we make use of semirings to allow different forms of quantitative reasoning uniformly. Semirings are most easily defined by starting out with monoids.

Definition 3 (Monoid)**.** *A* monoid $\mathcal{M} = (M, \otimes, e_\otimes)$ *consists of an associative* binary operation $\otimes$ *on a set* M *with* neutral element $e_\otimes$*, also called* identity element*. Here, a binary operation on* M *is a function* $\otimes : M \times M \to M$ *that maps pairs of values from* M *to a value in* M*. We write the application of such a binary operation* $\otimes$ *to a pair* (m_1, m_2) *of values* $m_1, m_2 \in M$ *in infix notation* $m_1 \otimes m_2$*.*

A value $e_\otimes \in M$ *is a neutral element for a binary operation* $\otimes$ *on* M *if for all values* $m \in M$ *it holds that*

$$e_\otimes \otimes m = m = m \otimes e_\otimes.$$

Additionally, a binary operation $\otimes$ *on* M *is*

- associative*, if for all* $m, m', m'' \in M$ *it holds that*

$$m \otimes (m' \otimes m'') = (m \otimes m') \otimes m'';$$

- commutative*, if for all* $m, m' \in M$ *it holds that*

$$m \otimes m' = m' \otimes m;$$

- idempotent*, if for all* $m \in M$ *it holds that*

$$m \otimes m = m;$$

- invertible*, if for all* $m \in M$ *there exists a unique* $m' \in M$ *such that*

$$m' \otimes m = e_\otimes = m \otimes m'.$$

A monoid is respectively commutative, idempotent, and invertible if its binary operation is.

Some examples of monoids are

- $\textsc{Strings} = (\{0,1\}^*, \odot, \varepsilon)$, the set of binary strings with concatenation $\odot$ and empty string ε is a non-commutative, non-idempotent, and non-invertible monoid,
- $\mathcal{P}(A) = (2^A, \cup, \emptyset)$, the set of subsets for a set A with union $\cup$ is a commutative, idempotent and non-invertible monoid,
- $\mathcal{P}(A) = (2^A, \cap, A)$, the set of subsets for a set A with intersection $\cap$ is a commutative, idempotent and non-invertible monoid,
- $\mathbb{Z} = (\mathbb{Z}, +, 0)$, the integers with addition $+$ is a commutative, non-idempotent and invertible monoid.

Based on monoids, we introduce semirings.

Definition 4 (Semiring)**.** *A* semiring $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ *is a nonempty set* R *equipped with two binary operations* $\oplus$ *and* $\otimes$*, called addition and multiplication, such that*

- $(R, \oplus)$ *is a commutative monoid with identity element* $e_\oplus$*,*
- $(R, \otimes)$ *is a monoid with identity element* $e_\otimes$*,*
- *multiplication left and right distributes over addition, i.e., for all* $r, r', r'' \in R$ *it holds that*

$$r \otimes (r' \oplus r'') = r \otimes r' \oplus r \otimes r''$$
$$(r' \oplus r'') \otimes r = r' \otimes r \oplus r'' \otimes r$$

- *and multiplication by* $e_\oplus$ *annihilates* R*, i.e., for all* $r \in R$ *it holds that*

$$r \otimes e_\oplus = e_\oplus = e_\oplus \otimes r.$$

Furthermore, $\mathcal{R}$ *is*

- commutative*, if* $(R, \otimes)$ *is commutative, and*
- idempotent*, if* $(R, \oplus)$ *is idempotent.*

If $(R, \oplus)$ *or* $(R, \otimes)$ *are invertible we define their inverse functions*

$$-(\cdot) : R \to R \qquad \text{such that} \qquad r \oplus -r = e_\oplus$$
$$\text{resp. } (\cdot)^{-1} : R \setminus \{e_\oplus\} \to R \qquad \text{such that} \qquad r \otimes r^{-1} = e_\otimes$$

Some examples of semirings are

- $\mathbb{B} = (\{0,1\}, \vee, \wedge, 0, 1)$, the Boolean semiring, with disjunction and conjunction as addition and multiplication,
- $\mathbb{F} = (\mathbb{F}, +, \cdot, 0, 1)$, for $\mathbb{F} \in \{\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}\}$ the semiring of the numbers in $\mathbb{F}$ with addition and multiplication,
- $\mathbb{N}_\infty = (\mathbb{N} \cup \{\infty\}, +, \cdot, 0, 1)$, the extension of the natural numbers with infinity, where $\infty + n = \infty$ and $\infty \cdot m = \infty$, for $m \neq 0$,
- $\mathcal{P}(A) = (2^A, \cup, \cap, \emptyset, A)$, the semiring over the powerset of A with union and intersection,
- $\mathcal{R}_{\max,+} = (\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$, the max-plus semiring,
- $\mathcal{R}_{\min,+} = (\mathbb{N} \cup \{\infty\}, \min, +, \infty, 0)$, the min-plus semiring,
- $\mathcal{R}_{\max,\cdot} = (\mathbb{N} \cup \{-\infty\}, \max, \cdot, -\infty, 0)$, the max-times semiring,
- $\mathcal{R}_{\min,\cdot} = (\mathbb{N} \cup \{\infty\}, \min, \cdot, \infty, 0)$, the min-times semiring,
- $\mathrm{GRAD} = (\{(v,u) \mid v, u \in \mathbb{R}\}, +, \otimes, (0,0), (1,0))$, the gradient/expectation semiring [Eis02], where addition is coordinate-wise and

$$(a_1, b_1) \otimes (a_2, b_2) = (a_1 \cdot a_2, a_2 \cdot b_1 + a_1 \cdot b_2),$$

- $\mathcal{P} = ([0,1], +, \cdot, 0, 1)$, the probability semiring,
- $\mathcal{R}[(x_i)_\alpha] = (R[(x_i)_\alpha], \oplus, \otimes, e_\oplus, e_\otimes)$, for $\alpha \in \mathbb{N}$ (resp. $\alpha = \infty$), is the semiring of polynomials with variables $x_1, \ldots, x_\alpha$ (resp. $x_1, x_2, \ldots$) and coefficients from the semiring $\mathcal{R}$, and
- $\mathbb{T} = (\{e\}, \odot, \odot, e, e)$, the trivial semiring, where $e \odot e = e$.

Another list of semirings, which is annotated with applications, can be found in [KVD17].

In order to connect the quantitative aspects of semirings and the qualitative ones of logics we use weighted logics. They were initially introduced by Droste and Gastin [DG07] in the second order setting. Here, we only introduce the restricted version for propositional logic.

Definition 5 (Syntax)**.** *Let $\mathcal{V}$ be a set of propositional variables and let $\mathcal{R}$ be a semiring of the form $(R, \oplus, \otimes, e_\oplus, e_\otimes)$. A* weighted *(propositional) formula over $\mathcal{R}$ is of the form α given by the grammar*

$$\alpha ::= k \mid v \mid \neg v \mid \alpha + \alpha \mid \alpha * \alpha$$

where $k \in R$ and $v \in \mathcal{V}$.

We can evaluate weighted formulas with respect to an interpretation to obtain a value from the semiring.

Definition 6 (Semantics)**.** *Given a weighted propositional formula α over a semiring $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ and propositional variables, i.e., nullary predicates, from $\mathcal{V}$ as well as an interpretation $\mathcal{I} \subseteq \mathbf{HB}(\mathcal{V}, \emptyset)$, the semantics $[\![\alpha]\!]_\mathcal{R}(\mathcal{I})$ of α over $\mathcal{R}$ w.r.t. $\mathcal{I}$ is defined as follows:*

$$
\begin{aligned}
[\![k]\!]_\mathcal{R}(\mathcal{I}) &= k \\
[\![v]\!]_\mathcal{R}(\mathcal{I}) &= \begin{cases} e_\otimes & v \in \mathcal{I} \\ e_\oplus & \text{otherwise.} \end{cases} \quad (v \in \mathcal{V}) \\
[\![\neg v]\!]_\mathcal{R}(\mathcal{I}) &= \begin{cases} e_\oplus & v \in \mathcal{I} \\ e_\otimes & \text{otherwise.} \end{cases} \quad (v \in \mathcal{V}) \\
[\![\alpha_1 + \alpha_2]\!]_\mathcal{R}(\mathcal{I}) &= [\![\alpha_1]\!]_\mathcal{R}(\mathcal{I}) \oplus [\![\alpha_2]\!]_\mathcal{R}(\mathcal{I}) \\
[\![\alpha_1 * \alpha_2]\!]_\mathcal{R}(\mathcal{I}) &= [\![\alpha_1]\!]_\mathcal{R}(\mathcal{I}) \otimes [\![\alpha_2]\!]_\mathcal{R}(\mathcal{I}).
\end{aligned}
$$

Example 3 (Weighted Formula)**.** *An example of a weighted formula is*

$$\alpha = \mathit{reach}(\mathit{bsgasse}) + \mathit{reach}(\mathit{kplatz}) + \mathit{reach}(\mathit{tsgasse}).$$

Intuitively, α interpreted over $\mathbb{N}$ counts how many stations we reach under a given assignment. Accordingly, its semantics with respect to the interpretation

$$\mathcal{I} = \{\mathit{reach}(\mathit{bsgasse}), \mathit{reach}(\mathit{tsgasse})\}$$

is given by

$$
\begin{aligned}
[\![\alpha]\!]_\mathbb{N}(\mathcal{I}) &= [\![\mathit{reach}(\mathit{bsgasse}) + \mathit{reach}(\mathit{kplatz}) + \mathit{reach}(\mathit{tsgasse})]\!]_\mathbb{N}(\mathcal{I}) \\
&= [\![\mathit{reach}(\mathit{bsgasse})]\!]_\mathbb{N}(\mathcal{I}) + [\![\mathit{reach}(\mathit{kplatz})]\!]_\mathbb{N}(\mathcal{I}) + [\![\mathit{reach}(\mathit{tsgasse})]\!]_\mathbb{N}(\mathcal{I}) \\
&= 1 + 0 + 1 = 2,
\end{aligned}
$$

as expected.

However, we can not only interpret α over the semiring $\mathbb{N}$. Over another semiring, we just might get a different result. For example, over the trivial semiring $\mathbb{T}$, we get the unique semiring value e as a result. Over the Boolean semiring $\mathbb{B}$, α tells us whether we reach any of the stations, since addition $+$ corresponds to disjunction and $e_\oplus$ is falsum and $e_\otimes$ is truth. Thus, over $\mathbb{B}$ for the same interpretation $\mathcal{I}$ we obtain

$$[\![\alpha]\!]_\mathbb{B}(\mathcal{I}) = 1.$$

With this in mind, we can begin introducing our own extensions of answer set programming.

2.2 Model Level Quantitative Reasoning

The goal of this section is to find a general framework for quantitative reasoning over the set of solutions in the context of ASP. That is, we want to be able to model

- probabilistic reasoning,
- preferential reasoning,
- quantitative queries, and
- (weighted) model counting.

Furthermore, we want to achieve this in a uniform manner.

In the following, we define *algebraic measures*, an extension of ASP that has the desired properties. After the introduction, we compare it to other quantitative extensions of ASP, for probabilistic reasoning, preferential reasoning, and weighted model counting to show that algebraic measures are indeed not only a uniform framework but also general enough.

2.2.1 Algebraic Measures

We define algebraic measures to combine the qualitative language of answer set programs with the quantitative one of weighted logic.

Definition 7 (Algebraic Measure)**.** *An* algebraic measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$ *consists of an answer set program* Π*, a weighted formula* α*, and a semiring* $\mathcal{R}$*. Then, the weight of an answer set* $\mathcal{I} \in \mathcal{AS}(\Pi)$ *under* μ *is defined by*

$$\mu(\mathcal{I}) = [\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}).$$

Additionally, the result of an (atomic) query *for an atom* $a \in \mathbf{HB}(\Pi)$ *is given by*

$$\mu(a) = \bigoplus_{\mathcal{I} \in \mathcal{AS}(\Pi), a \in \mathcal{I}} \mu(\mathcal{I}),$$

and the result of the overall weight query *of* Π *is*

$$\mu(\Pi) = \bigoplus_{\mathcal{I} \in \mathcal{AS}(\Pi)} \mu(\mathcal{I}).$$

Intuitively, the idea here is that for an algebraic measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$ the semiring $\mathcal{R}$ determines the mode of quantitative computation, Π states the logical background theory that determines which interpretations are solutions and α assigns each answer set $\mathcal{I}$ a weight over the semiring, by performing a calculation over the semiring that depends on the satisfaction of atomic formulas in the interpretation $\mathcal{I}$.

Then, evaluating $\mu(\mathcal{I})$ for an answer set $\mathcal{I}$ allows us to perform quantitative queries. Evaluating an atomic query $\mu(a)$ for an atomic formula $a \in \mathbf{HB}(\Pi)$ allows us to query

for the aggregated weight of the answer sets that satisfy a. This enable for example probabilistic inference, if μ is a probability measure, i.e., if $\mu(\Pi) = 1$ and $\mathcal{R}$ is the probabilistic semiring $\mathcal{P}$. In this setting, the query $\mu(a)$ gives us the probability that a is true in an answer set chosen according to the probability distribution defined by μ.

Last but not least, overall weight queries aggregate the weights of all answer sets, without a query atom that needs to be satisfied. If μ is a probability measure, the overall weight is not particularly interesting, as it is always one. However, over other semirings such as $\mathbb{N}$, $\mathcal{R}_{\max,+}$ and GRAD the overall weight can provide us with interesting information such as the number of answer sets, the weight of an optimal answer set or an expected value, respectively.

We consider how each of the queries with algebraic measures can be used in the context of our running example.

Example 4 (cont.)**.** *First, we consider quantitative queries over a single answer set. For example, we can ask the question how many changes a given journey uses. For this, let $\mathcal{D}$ denote the domain of our program. Then we can use the following weighted formula*

$$\alpha_{change} = -1 + \Sigma_{m,s \in \mathcal{D}} enter(m, s),$$

which counts the number of times we enter a metro and subtracts one. If we use the measure $\mu_{change} = \langle \Pi_{in} \cup \Pi_m, \alpha_{change}, \mathbb{N} \rangle$ and evaluate it under the answer set $\mathcal{I}_{initial}$ from the previous example, we obtain

$$\begin{aligned} \mu_{change}(\mathcal{I}_{initial}) &= [\![-1 + \Sigma_{m,s \in \mathcal{D}} enter(m, s)]\!]_{\mathbb{N}}(\mathcal{I}_{inital}) \\ &= -1 + \sum_{m,s \in \mathcal{D}} \begin{cases} 1 & \text{if } enter(m, s) \in \mathcal{I}_{inital}, \\ 0 & \text{otherwise.} \end{cases} \\ &= -1 + 3 = 2. \end{aligned}$$

Second, we are interested in the probability of starting our journey with the "U4" metro, if we select one answer set as a journey uniformly at random. Note that for the current input facts Π_{in}, this question can be answered very easily, since there is only one possible journey. Assume therefore that we replace Π_{in} by $\Pi_{complete}$, which contains the complete information about the Viennese public transport network. In either case, we can use an overall weight query in combination with an atomic query to obtain the result using the measure $\mu_{count} = \langle \Pi_m \cup \Pi_{complete}, 1, \mathbb{N} \rangle$. Namely, we can compute $\mu_{count}(enter(\text{"}U4\text{"}, bsgasse))$ the atomic query for $enter(\text{"}U4\text{"}, bsgasse)$ and the overall weight $\mu_{count}(\Pi_m \cup \Pi_{complete})$. The former gives us the number of journeys in which we start with the U4 metro, the latter gives us the total number of journeys. By computing their quotient, we thus obtain the probability of starting our journey with the U4 metro, if we choose a possible journey uniformly at random.

As a side product of the previous example, we observe that in order to count the answer sets of a program we can perform an overall weight query using the weighted formula 1 over the semiring of the natural numbers.

For more details on the possible applications of algebraic measures we refer the interested reader to Section 4.3.

Having introduced algebraic measures, we can consider their expressivity in comparison to other formalisms.

2.2.2 Relation to Similar Formalisms

As stated before the goal of the definition of algebraic measures is to introduce a general quantitative framework that captures many of the different relevant quantitative extensions of answer set programming and, furthermore, does so in a uniform manner. The uniformity is given as a consequence of the parameterization of the definition with semirings, which allows for a wide range of quantitative computations by using the appropriate semiring. As for the generality, we investigate it in this subsection by taking a close look at the expressivity of algebraic measures compared to other quantitative extensions of ASP.

In our comparison, we include the related works of

- LP^{MLN} [LY17];
- P-log [BGR09];
- ProbLog [DKT07] and its generalization to algebraic Prolog [KVD11];
- weak constraints [BLR00];
- asprin [Bre+15];
- and others [NM15].

In the following we compare them in the order specified in the itemization. Apart from that, we give an overlook of other formalisms that we did not consider in detail and the expected relation between them and algebraic measures. Finally, we discuss whether the expressiveness of algebraic measures suffices for our needs, taking into account their relation to the other quantitative reasoning extensions we considered.

LP^{MLN} LP^{MLN} concerns itself with probabilistic logic programming, more precisely in LP^{MLN} we assign interpretations a probability of being correct. For this, we consider a set of weighted rules

$$\Pi = \{w_1 : r_1, \ldots, w_n : r_n\}$$

where $w_i \in \mathbb{R}$ or w_i is the variable x and r_i a rule. Note that we restrict ourselves to normal rules here for sake of simplicity, whereas the definition in [LW16] also allows disjunctions in the head. All the results hold for disjunctive rules in the same manner though, if we allow disjunctive answer set programs in algebraic measures too.

The weight scheme is the same as that of Markov Logic, i.e., the probability of a rule being correct is log-linear in its weight, i.e., the logarithm of the probability grows linearly with respect to the weight. If $w_i = x$, the weight is interpreted to be arbitrarily large [LY17]. Due to this, the definition of answer sets is changed in such a manner that we can also have answer sets that do not satisfy all rules of an LP$^{\text{MLN}}$ program. However, we still need stability on the rules they satisfy. Therefore, the reduct of an LP$^{\text{MLN}}$ program with respect to an interpretation $\mathcal{I}$ is defined as

$$\Pi_{\mathcal{I}} = \{r \mid w : r \in \Pi, \mathcal{I} \models r\}.$$

The set $\mathbf{SM}(\Pi)$ of models of an LP$^{\text{MLN}}$ program Π is then defined as

$$\mathbf{SM}(\Pi) = \{\mathcal{I} \subseteq \mathbf{HB}(\Pi) \mid \mathcal{I} \text{ is a (classical) answer set of } \Pi_{\mathcal{I}}\}.$$

With this in mind we can formally define the weight of an interpretation with respect to Π.

Definition 8 (Weight Function W_Π, Probability)**.** *Let*

$$\Pi = \{w_1 : r_1, \ldots, w_n : r_n\}$$

be an LP^{MLN} program and let $\mathcal{I} \subseteq \mathbf{HB}(\Pi)$ be an interpretation of Π. Then the weight of $\mathcal{I}$ with respect to Π in dependence on x, where x is the unique variable that occurs in place of weights w_i, is defined as

$$W_\Pi(\mathcal{I}) = \exp\left(\sum\{w \mid w : r \in \Pi, \mathcal{I} \models r\}\right),$$

if $\mathcal{I} \in \mathbf{SM}(\Pi)$ and 0, otherwise.

Furthermore, the probability of an interpretation $\mathcal{I}$ is defined as

$$Pr_\Pi(\mathcal{I}) = \begin{cases} \lim\limits_{x\to\infty} \dfrac{W_\Pi(\mathcal{I})}{\sum_{\mathcal{J}\in\mathbf{SM}(\Pi)} W_\Pi(\mathcal{J})} & \mathcal{I} \in \mathbf{SM}(\Pi), \\ 0 & \textit{otherwise.} \end{cases}$$

Note that in the definition of W_Π the value W_Π is still dependent on the variable x. Since the meaning of $x : r$ is that the weight of the rule r is arbitrarily large, we apply the limit in the definition of the probability. This way, we obtain a real valued number.

The above can equivalently be modeled as an algebraic measure $\mu = \langle \Pi', \alpha, \mathcal{R} \rangle$, where

$$\begin{aligned} \Pi' =& \{head(r) \leftarrow body(r), aux_r \mid w : r \in \Pi\} && (2.10)\\ &\cup \{\leftarrow head(r), \text{not } aux_r \mid w : r \in \Pi\} && (2.11)\\ &\cup \{\leftarrow \text{not } b, \text{not } aux_r \mid w : r \in \Pi, b \in body(r)\} && (2.12)\\ &\cup \{\{aux_r\} \leftarrow \mid w : r \in \Pi\}, && (2.13)\\ \alpha =& \Pi_{w:r\in\Pi, w\neq x} \exp(w) * aux_r * \Pi_{w:r\in\Pi, w=x} x * aux_r, \text{ and} && (2.14) \end{aligned}$$

$$\mathcal{R} = \mathbb{R}[x]. \tag{2.15}$$

The idea here is that we condition all rules on a new auxiliary choice atom aux_r. This choice atom intuitively chooses whether the rule r has to be satisfied. In order to be in line with the LP$^{\text{MLN}}$ semantics, we furthermore require that when a rule r is satisfied, then we also need to include it in our program, meaning the choice atom aux_r needs to be true.

Note that also here, the values of the algebraic measure are over $\mathbb{R}[x]$ and therefore depend on x. Contrary to W_Π, the weighted formula α does not refer to $\exp(x)$ but to x. However, while this makes a difference when comparing $W_\Pi(\mathcal{I})$ and $\mu(\mathcal{I})$, it does not for the final probability measure.

Theorem 9. *Let Π be an LP^{MLN} program, let $\mathcal{I} \subseteq \mathbf{HB}(\Pi)$ be an interpretation of Π, and let $comp(\mathcal{I}) = \mathcal{I} \cup \{aux_r \mid \mathcal{I} \models r\}$. Then it holds that*

1. *$\mathcal{I} \in \mathbf{SM}(\Pi)$ iff $comp(\mathcal{I}) \in \mathcal{AS}(\Pi')$;*
2. *$W_\Pi(\mathcal{I}) = \mu(comp(\mathcal{I}))[x/\exp(x)]$, where $[x/\exp(x)]$ denotes that every occurrence of x is replaced by $\exp(x)$; and*
3. $$\lim_{x\to\infty} \frac{\mu(comp(\mathcal{I}))}{\mu(\Pi)} = \begin{cases} \lim\limits_{x\to\infty} \dfrac{W_\Pi(\mathcal{I})}{\sum_{\mathcal{J}\in\mathbf{SM}(\Pi)} W_\Pi(\mathcal{J})} & \mathcal{I} \in \mathbf{SM}(\Pi), \\ 0 & \textit{otherwise.} \end{cases}.$$

Proof. We start with 1. and consider an interpretation $\mathcal{I}$. From the rules of the form (2.11), (2.12) in the definition of Π' it follows that if aux_r is not included in an interpretation $\mathcal{I}^*$ of Π', then the rule r cannot be satisfied. On the other hand, from the rules of the form (2.10) it follows that if aux_r is included in an interpretation $\mathcal{I}^*$ of Π', then the rule r must be satisfied. Therefore, for the interpretations of $\mathcal{I}^*$ of Π such that $\mathcal{I}^* \cap \{aux_r \mid r \in \Pi\} = \text{comp}(\mathcal{I}) \setminus \mathcal{I}$ it follows that $\mathcal{I}^*$ is an answer set of Π' iff $\mathcal{I}^* \setminus \{aux_r \mid r \in \Pi\}$ is an answer set of $\{r \mid r \in \Pi, aux_r \in \mathcal{I}^*\}$. Since the interpretation $\text{comp}(\mathcal{I})$ contains exactly those atoms of the form aux_r such that r is satisfied by $\mathcal{I}$ the program $\{r \mid r \in \Pi, aux_r \in \mathcal{I}^*\}$ is equal to the reduct $\Pi_\mathcal{I}$. So, it holds that $\mathcal{I}$ is an answer set of $\Pi_\mathcal{I}$ iff $\text{comp}(\mathcal{I})$ is an answer set of Π'.

For 2. recall that

$$W_\Pi(\mathcal{I}) = \exp\left(\sum\{w \mid w : r \in \Pi, \mathcal{I} \models r\}\right),$$

if $\mathcal{I} \in \mathbf{SM}(\Pi)$ and $W_\Pi(\mathcal{I}) = 0$, otherwise. Second, recall that also $\mu(\text{comp}(\mathcal{I}))$ is zero if $\text{comp}(\mathcal{I})$ is not an answer set of Π', which is the case iff $\mathcal{I} \notin \mathbf{SM}(\Pi)$ according to 1. Thus, it only remains to verify that $\mu(\text{comp}(\mathcal{I}))$ takes the correct value when $\text{comp}(\mathcal{I})$ is an answer set. This is easy to see:

$$\mu(\text{comp}(\mathcal{I})) = [\![\alpha]\!]_{\mathbb{R}[x]}(\text{comp}(\mathcal{I}))$$

$$= [\![\Pi_{w:r\in\Pi,w\neq x} \exp(w) * aux_r * \Pi_{w:r\in\Pi,w=x} x * aux_r]\!]_{\mathbb{R}[x]}(\mathrm{comp}(\mathcal{I}))$$
$$= [\![\Pi_{w:r\in\Pi,w\neq x} \exp(w) * aux_r * \Pi_{w:r\in\Pi,w=x} x * aux_r]\!]_{\mathbb{R}[x]}(\mathcal{I} \cup \{aux_r \mid \mathcal{I} \models r\})$$
$$= \prod_{w:r\in\Pi|\mathcal{I}\models r,w\neq x} \exp(w) \cdot \prod_{w:r\in\Pi|\mathcal{I}\models r,w=x} x$$

By replacing x by $\exp(x)$ we get

$$\exp\left(\sum\{w \mid w : r \in \Pi \mid \mathcal{I} \models r, w \neq x\} + \sum\{x \mid x : r \in \Pi \mid \mathcal{I} \models r\}\right),$$

which is equal to $W_\Pi(\mathcal{I})$ as desired.

Last but not least, for 3. we first observe that from 1. and 2. it follows that

$$\mu(\Pi)[x/\exp(x)] = \sum_{\mathcal{J}\in\mathbf{SM}(\Pi)} W_\Pi(\mathcal{J}).$$

Furthermore, from

$$\lim_{x\to\infty} \frac{\mu(\mathrm{comp}(\mathcal{I}))}{\mu(\Pi)} = \lim_{\log(x)\to\infty} \frac{\mu(\mathrm{comp}(\mathcal{I}))}{\mu(\Pi)} = \lim_{x\to\infty} \frac{\mu(\mathrm{comp}(\mathcal{I}))[x/\exp(x)]}{\mu(\Pi)[x/\exp(x)]}$$

the previous observation and 2. it follows that the claim holds. □

Overall, we see that algebraic measures subsume $\mathrm{LP}^{\mathrm{MLN}}$ on a very high level: not only can we express all probability distributions that can be expressed using $\mathrm{LP}^{\mathrm{MLN}}$, but we can even capture the inner workings of its definitions. That is, (i) we can define an algebraic measure over the polynomial semiring $\mathbb{R}[x]$ to model the weight function W_Π and (ii) we can define it without the need for an adapted answer set semantics - which $\mathrm{LP}^{\mathrm{MLN}}$ needs - by exploiting the intertwinedness of qualitative and quantitative reasoning that is possible with algebraic measures.

P-log

In the probabilistic extension of logic programming by Baral, Gelfond, and Rushton [BGR09] answer sets are assigned a probability, leading to the possibility do draw probabilistic inferences. Here, one can specify that an attribute $a(\bar{t})$ has a random value X from $p(X)$, i.e., the domain of values that satisfy the predicate p. This can further be conditioned on the truth of a set of literals B. Such probabilistic choice rules are specified in P-log syntax using a rule of the form

$$[r]random(a(\bar{t}) : \{X : p(X)\}) \leftarrow B.$$

Furthermore, it is possible to model the probability that a random attribute $a(\bar{t})$ has a specific value y, when generated by rule r, given that a set B' of literals is satisfied, using

$$pr_r(a(\bar{t}) = y \mid B') = v.$$

The remainder $1 - P$, where P is the sum of the already assigned probabilities, is distributed uniformly among the possible values y whose probability is unspecified. More details can be found in [BGR09].

Regarding the relation to algebraic measures, we can rely on existing results. In [LY17] it was shown that LP^{MLN} subsumes P-log. Thus, since algebraic measures subsume LP^{MLN}, also algebraic measures subsume P-log.

ProbLog and algebraic Prolog

In ProbLog, one is interested in the probability of a query succeeding, given a set of probabilistic rules

$$\Pi = \{p_1 : c_1, \ldots, p_n : c_n\}$$

where c_i is a ProbLog rule and p_i is the probability of being correct. ProbLog rules are rules, that are either of the form

$$a \leftarrow \tag{2.16}$$

or of the form

$$a \leftarrow b_1, \ldots, b_n, \backslash + c_1, \ldots, \backslash + c_m, \tag{2.17}$$

where $c_1, \ldots, c_n$ occur in a rule of the form 2.16.

The probability of a query q, where $q \in \mathbf{HB}(\Pi)$ is an atom, is then given by

$$P(q \mid \Pi) = \sum_{\Pi' \subseteq \Pi} \prod_{p:c \in \Pi'} p \prod_{p:c \in \Pi \setminus \Pi'} (1-p) \cdot \begin{cases} 1 & \Pi' \models q, \\ 0 & \text{otherwise.} \end{cases}.$$

Again, we can rely on existing results for the expressiveness comparison between ProbLog and algebraic measures. In [LY17] it was shown that LP^{MLN} subsumes ProbLog. Thus, since algebraic measures subsume LP^{MLN}, also algebraic measures subsume ProbLog.

The current results for the expressiveness do not cover algebraic Prolog [KVD11] though, which is very similar to ProbLog and can in fact be seen as a generalization of ProbLog over general semirings.

The definition of algebraic Prolog is actually very similar to that of algebraic measures, in the sense that quantitative and qualitative reasoning are somewhat separated. An algebraic Prolog program [KVD11] consists of:

- a commutative semiring $(A, \oplus, \otimes, e_\oplus, e_\otimes)$,
- a finite set of ground algebraic facts $F = \{f_1, \ldots, f_n\}$,
- a finite set BK of ProbLog rules, called background knowledge, and

- a labeling function $\alpha : L(F) \to A$ that labels literals from

$$L(F) = \{a \mid a \in F\} \cup \{\neg a \mid a \in F\}$$

to a semiring value in A.

Different from ProbLog programs, where all rules have a label, only the algebraic facts have a label here and the rest of the rules are to be seen as strict, i.e., always true. Note that this implies that for each interpretation $\mathcal{I}$ of the algebraic facts in F the program $BK \cup \mathcal{I}$ has exactly one answer set.

The label $A(\mathcal{I})$ of an interpretation $\mathcal{I}$ is thus not defined for interpretations over all atoms $\mathbf{HB}(BK)$ but for interpretations of the algebraic facts in F. Namely,

$$A(\mathcal{I}) = \bigotimes_{a \in \mathcal{I}} \alpha(a) \otimes \bigotimes_{a \in F \setminus \mathcal{I}} \alpha(\neg a).$$

This is lifted to query atoms $q \in \mathbf{HB}(BK)$ by defining

$$A(q) = \bigoplus \{A(\mathcal{I}) \mid \mathcal{I} \subseteq F, BK \cup \mathcal{I} \models q\}.$$

We can model queries for labels of atoms with respect to an algebraic Prolog program also using algebraic measures. Namely, given an algebraic Prolog program $\langle (A, \oplus, \otimes, e_\oplus, e_\otimes), F, BK, \alpha \rangle$, we define $\mu = \langle \Pi', \alpha', (A, \oplus, \otimes, e_\oplus, e_\otimes) \rangle$ as the algebraic measure, where

$$\Pi' = \{\{a\} \leftarrow \mid a \in F\} \cup BK, \text{ and}$$
$$\alpha' = \Pi_{a \in F}(a * \alpha(a) + \neg a * \alpha(\neg a)).$$

Then the following holds:

Corollary 10. *Given the above definition,*

- *for each interpretation $\mathcal{I} \subseteq F$ it holds that $A(\mathcal{I}) = \mu(comp(\mathcal{I}))$, and*
- *for each atom $q \in \mathbf{HB}(BK)$ it holds that $A(q) = \mu(q)$.*

where $comp(\mathcal{I}) = \{a \in \mathbf{HB}(BK) \mid BK \cup \mathcal{I} \models a\}$.

Weak Constraints

The idea of weak constraints [BLR00] is to an extend answer set program Π by a set C of constraints of the form

$$: \sim b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m. \qquad [w],$$

where $b_1, \ldots, b_n$ and $c_1, \ldots, c_m$ are atoms and w is an integer weight specifying how (un)desirable it is to satisfy the constraint. Additionally, we may in practice specify weak

constraints with a priority level p, by replacing w with $w@p$, however, since it has been shown that we can assume without loss of generality that the program is given without priority levels, we restrict ourselves to this case.

The preferred answer sets of a program Π with respect to a set C of weak constraints is then given by those answer sets that satisfy the most weak constraints, when counted respectively to their weight w. Formally,

Definition 11. *The* penalty *of an answer set $\mathcal{I} \in \mathcal{AS}(\Pi)$ is defined as*

$$p_C(\mathcal{I}) = \sum_{:\sim B(r).[w] \in N(C,\mathcal{I})} w,$$

where

$$N(C,\mathcal{I}) = \{: \sim b_1, \ldots, b_n, not\, c_1, \ldots, not\, c_m.[w] \in C \mid b_1, \ldots, b_n \in \mathcal{I}, c_1, \ldots, c_m \notin \mathcal{I}\}$$

is the set of non-satisfied weak constraints.

Then an answer set $\mathcal{I} \in \mathcal{AS}(\Pi)$ is preferred*, if there does not exist an answer set $\mathcal{I}' \in \mathcal{AS}(\Pi)$ with a strictly lower penalty, i.e. with $p_C(\mathcal{I}) > p_C(\mathcal{I}')$.*

We can express the optimization problem associated with programs with weak constraints also with algebraic measures. Note that this already follows from results in [LY17] together with Theorem 9, since it was shown that LP^{MLN} programs can also model the penalty function specified by weak constraints. However, we can do even more: apart form specifying the penalty function, we can also use overall weight queries in order to compute preferred answer sets.

In a first step, we define the penalty function $p_C(.)$ by means of an algebraic measure. For this, we make use of the min-plus semiring $\mathcal{R}_{\min,+}$. Given a set C of weak constraints, we can construct a corresponding weighted formula α_C over $\mathcal{R}_{\min,+}$ as

$$\alpha_C = \Pi_{:\sim B(r).[w] \in C} \alpha_{B(r),w}, \text{ where}$$
$$\alpha_{b_1,\ldots,b_n,\text{not } c_1,\ldots,\text{not } c_m,w} = w * \Pi_{i=1}^{n} b_i * \Pi_{j=1}^{m} \neg c_j + \Sigma_{i=1}^{n} \neg b_i + \Sigma_{j=1}^{m} c_j$$

Using this weighted formula, we get the following result:

Corollary 12. *Given the above definition of the weighted formula α_C, it holds for $\mu = \langle \Pi, \alpha_C, \mathcal{R}_{\min,+} \rangle$ that*

- *for any answer set $\mathcal{I}$ of Π the penalty $p_C(\mathcal{I})$ is equal to $\mu(\mathcal{I})$; and*
- *the overall weight of μ is*

$$\mu(\Pi) = \bigoplus_{\mathcal{I} \in \mathcal{AS}(\Pi)} \mu(\mathcal{I}) = \min_{\mathcal{I} \in \mathcal{AS}(\Pi)} \mu(\mathcal{I}) = \min_{\mathcal{I} \in \mathcal{AS}(\Pi)} p_C(\mathcal{I}),$$

i.e. the minimum penalty that can be achieved.

Arguably, computing the optimal penalty value is a different problem, from computing a preferred answer set of a program and a set of weak constraints. However, assuming w.l.o.g. that our Herbrand base $\mathbf{HB}(\Pi)$ is finite, we can also achieve the latter by using a different weighted formula

$$\begin{aligned}\alpha_C^M &= \alpha_{C\cdot 2^{|\mathbf{HB}(\Pi)|}} * \alpha^M \\ \alpha_{C\cdot 2^{|\mathbf{HB}(\Pi)|}} &= \Pi_{:\sim B(r).[w]\in C} \alpha_{B(r),w\cdot 2^{|\mathbf{HB}(\Pi)|}} \\ \alpha^M &= \Sigma_{a\in\mathbf{HB}(\Pi)} 2^{\mathrm{idx}(a)} * a + \neg a\end{aligned}$$

Here, $\mathrm{idx}(a)$ assigns each atom $a \in \mathbf{HB}(\Pi)$ a unique index from $\{0, \ldots, |\mathbf{HB}(\Pi)| - 1\}$.

The idea behind the weighted formula α_C^M is that we split the weight it computes into two parts, one for the penalty and one for the assignment that was used. This is achieved by multiplying every local penalty w of every weak constraint by $2^{|\mathbf{HB}(\Pi)|}$ and by intuitively adding a penalty of $2^{\mathrm{idx}(a)}$ when the atom a holds. On the other hand, when the atom a does not hold and $\neg a$ is satisfied, we add the neutral element for the multiplication of $\mathcal{R}_{\min,+}$. Since the multiplication of $\mathcal{R}_{\min,+}$ is $+$, the neutral element is 0, meaning we do not change the penalty of $\neg a$ is satisfied. Due to the factor $2^{|\mathbf{HB}(\Pi)|}$, the original penalties and the penalties for the atoms do not interact and we obtain the following result:

Lemma 13 (Preferred Answer Set as Overall Weight)**.** *Given a program Π with a finite Herbrand base and weak constraints C, the overall weight of the algebraic measure $\mu = \langle \Pi, \alpha_C^M, \mathcal{R}_{\min,+} \rangle$ is equal to $2^{|\mathbf{HB}(\Pi)|} \cdot p + b$, where*

- *p is the optimal penalty and*
- *b is an integer smaller than $2^{|\mathbf{HB}(\Pi)|}$*
- *$b = \sum_{a\in\mathcal{I}_{\min}} 2^{idx(a)}$, where $\mathcal{I}_{\min}$ is the minimum lexicographical model (when the atoms are indexed by $idx(.)$) that is preferred with respect to C.*

This means that we can obtain the minimum lexicographical preferred answer set of a program Π with weak constraints C by computing the overall weight of an algebraic measure. We see that algebraic measures are not only suitable for the computation of the optimal penalty value but also for the computation of preferred models themselves. The use of algebraic measures for preferential reasoning is considered in more detail in [BEK21].

asprin

The asprin framework allows for the optimization of sets $\{w_i : \alpha_i \mid i = 1, \ldots, n\}$ of labeled formulae with respect to arbitrary aggregates of the labels w_i of the satisfied α_i; it subsumes most approaches for preferential reasoning in ASP [Bre+15]. We are bound to aggregates expressible by semiring operations; those mentioned in [Bre+15], viz. union of

sets, count of elements and sum of integers, are all captured by weighted formulae over the semirings $\mathcal{P}(A), \mathbb{N}$, and $\mathbb{Z}$ respectively. Using semirings also has benefits: it allows for complex expressions beyond an aggregation. In asprin, one can specify orders and define composite orders for multiple objectives. While algebraic measures can be used with any order, we do not provide a language for specifying and composing orders. A complete comparison is hard due to the short format of the conference paper [Bre+15], which does not admit space for all details.

Overall, asprin can handle any kind of preference, as long as the decision whether one answer set is better than another can be uniformly expressed as an answer set program. Algebraic measures, on the other hand, allow for the specification of measures that can be evaluated over an answer set. Given an order relation on the semiring values we can then check whether one answer set is preferred to another; however, the relation is not a necessary part of the algebraic measure. As we have seen in the previous subsubsection about weak constraints, we can sometimes compute a preferred answer set as the result of an overall weight query. It is, however, unclear whether such a strategy is feasible in general.

Others

There are further quantitative extensions for model level reasoning in the context of answer set programming besides the ones we discussed above. Some of them are not subsumed by algebraic measures.

One example is PrASP [NM14], which is a Nilsson-style probabilistic languages. Here, probabilities are specified in an indirect manner. That is, the probabilities are required to satisfy a set of (usually linear) inequalities. It follows, that instead of deriving exact values during probabilistic inference, we only obtain a range of possible values. Thus, it is unlikely that we can capture them with algebraic measures, where the specification is direct and probability values are uniquely determined.

General Annotated Logic Programs (GALP) [KS92] have multi-valued interpretations and allow for the specification of values using arbitrary monotone functions. Therefore, only fragments of algebraic measures can be expressed in GALP and vice versa.

Summary

We have seen that many quantitative extensions are subsumed by algebraic measures. Many probabilistic frameworks such as LP^{MLN}, P-log and ProbLog can be easily captured. Also optimization is possible and preferential reasoning with algebraic measures subsumes ASP extended with weak constraints and covers a considerable fragment of asprin. Also, highly general formalisms like algebraic Prolog, which themselves have been shown to be useful for a wide variety of different tasks, are subsumed by algebraic measures.

Notably, like algebraic Prolog, also algebraic measures tackle all of the above problems uniformly by a parameterization of the semantics with a semiring. Furthermore, by

having both an answer set program and a weighted formula, it is possible to separate the qualitative reasoning specification from the quantitative one. Algebraic measures distinguish themselves from algebraic Prolog programs, by allowing

- a richer fragment for the specification of the logical background theory, and
- weighted formulas, which can assign an interpretation a weight that stems from evaluating a complex expressions using both addition and multiplication of the semiring, instead of just a product of semiring values, as it is the case with algebraic Prolog's labeling function.

Overall, we argue that algebraic measures are a notable addition to the landscape of quantitative extensions of answer set programming as they are both highly general as well as uniform. As such they also fulfill the requirements we pose for an extension to be suitable for it to be lifted to the context of stream reasoning.

2.3 Truth Level Quantitative Reasoning

Apart from quantitative reasoning over the set of solutions, we also want to be able to express succinctly and easily which interpretations of a program are solutions. That is, we want to be able to use

- aggregates,
- weight constraints,
- arithmetic operations, and
- guessing in rule-heads.

As with the quantitative reasoning over the set of solutions, we want to achieve this in a uniform manner.

In the following, we will show that all of these constructs can be seen as quantitative constraints, again by introducing an algebraic semantics over semirings. For this, we define *algebraic constraints*, based on a generalization of first-order Here-and-There Logic over semirings. As in the previous section, this generalization allows us to specify weighted formulas over a semiring, whose value is computed using the semiring operations and depends on the given interpretation. However, contrary to the weighted formulas in the previous section, this dependence is in a sense "non-monotonic". This non-monotonicity comes from the fact that we generalized Here-and-There Logic, which is the basis of Equilibrium Logic [Pea06] that can be seen to capture the non-monotonicity of ASP.

We start off by giving the necessary preliminaries, namely sorted first-order Here-and-There Logic. Based on this, we introduce its weighted generalization over semirings

and define algebraic constraints as well as the syntax and semantics of programs that use algebraic constraints. After showing that these definitions satisfy some expected properties, we give an example use case that shows that the provenance semantics of Green, Karvounarakis, and Tannen [GKT07] can be computed with $\mathcal{AC}$-programs. Next, we consider properties such as safety, domain independence and strong equivalence in the context of $\mathcal{AC}$-programs, compare to other extensions of ASP that add constructs to enhance the succinctness of ASP as a language. Furthermore, we investigate the computational complexity of common reasoning tasks with $\mathcal{AC}$-programs. Finally, we conclude that algebraic constraints are indeed not only a uniform framework but also general enough.

2.3.1 Preliminaries

We start by introducing classical programs and their semantics. We use a variant of first-order HT semantics [PV08] as this facilitates the generalization of the semantics later on and is useful for work on strong equivalence. The variant is that we assign variables non-disjoint sorts, which lets us quantify over subsets of the domain.

We consider sorted first-order formulas over a *signature* $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$, where $\mathcal{D}$ is a set of domain elements, $\mathcal{P}$ is a set of predicates, $\mathcal{X}$ is a set of sorted variables, $\mathcal{S}$ is a set of sorts and $\mathfrak{r} : \mathcal{S} \to 2^{\mathcal{D}}$ is a range function assigning each sort a subset of the domain. When $x \in \mathcal{X}$, we write $s(x)$ for the sort of x. Then, *σ-formulas* are of the form

$$\phi ::= \bot \mid p(\overline{x}) \mid \phi \to \phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \exists y \phi \mid \forall y \phi, \tag{2.18}$$

where $p \in \mathcal{P}$, $\overline{x} = x_1, \ldots, x_n$, with $x_i \in \mathcal{D}$ or $x_i \in \mathcal{X}$ and $y \in \mathcal{X}$; $p(\overline{x})$ is called a *σ-atom*. We define $\neg\phi = \phi \to \bot$. A *σ-sentence* is a σ-formula without free variables.

Definition 14 (HT Semantics)**.** *Let $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ be a signature and $\mathcal{I}^H, \mathcal{I}^T$ be* σ-interpretations*, i.e. sets of σ-atoms without free variables over the predicates in $\mathcal{P}$ and elements in $\mathcal{D}$, s.t. $\mathcal{I}^H \subseteq \mathcal{I}^T$. Then $\mathcal{I} = (\mathcal{I}^H, \mathcal{I}^T)$ is a* σ-HT-interpretation *and $\mathcal{I}_w = (\mathcal{I}^H, \mathcal{I}^T, w)$, for $w \in \{H, T\}$, is a* pointed σ-HT-interpretation.

Satisfaction of a σ-sentence ϕ w.r.t. a pointed σ-HT-interpretation $\mathcal{I}_w = (\mathcal{I}^H, \mathcal{I}^T, w)$ is defined as follows, where we have the reflexive order $\geq$ on $\{H, T\}$, with $T \geq H$:

$$\begin{aligned}
&\mathcal{I}_w \not\models_\sigma \bot \\
&\mathcal{I}_w \models_\sigma p(\overline{x}) && \iff && p(\overline{x}) \in \mathcal{I}^w \\
&\mathcal{I}_w \models_\sigma \phi \to \psi && \iff && \mathcal{I}_{w'} \not\models_\sigma \phi \text{ or } \mathcal{I}_{w'} \models_\sigma \psi \text{ for all } w' \geq w \\
&\mathcal{I}_w \models_\sigma \phi \vee \psi && \iff && \mathcal{I}_w \models_\sigma \phi \text{ or } \mathcal{I}_w \models_\sigma \psi \\
&\mathcal{I}_w \models_\sigma \phi \wedge \psi && \iff && \mathcal{I}_w \models_\sigma \phi \text{ and } \mathcal{I}_w \models_\sigma \psi \\
&\mathcal{I}_w \models_\sigma \exists x \phi(x) && \iff && \mathcal{I}_w \models_\sigma \phi(\xi), \text{ for some } \xi \in \mathfrak{r}(s(x)) \\
&\mathcal{I}_w \models_\sigma \forall x \phi(x) && \iff && \mathcal{I}_w \models_\sigma \phi(\xi), \text{ for all } \xi \in \mathfrak{r}(s(x))
\end{aligned}$$

When T is a set of σ-sentences, then $\mathcal{I}_w \models_\sigma T$ if $\forall \phi \in T : \mathcal{I}_w \models_\sigma \phi$.

The semantics of classical rules and programs is introduced as an instantiation of the above semantics for restricted signatures. Let $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ be a *classical signature*, i.e. $\mathcal{S} = \{D\}, \mathfrak{r}(D) = \mathcal{D}$. Then a *rule* is of the form

$$r = H(r) \leftarrow B(r) = \phi \leftarrow \psi_1, \ldots, \psi_n, \neg\theta_1, \ldots, \neg\theta_m,$$

where ϕ, ψ_i, θ_j are σ-atoms, with free variables $x_1, \ldots, x_k \in \mathcal{X}$. Its semantics is that of the *universal closure*, which is the σ-formula $\forall x_1, \ldots, x_k \; B_\wedge(r) \rightarrow \phi$ where $B_\wedge(r)$ is $\psi_1 \wedge \cdots \wedge \psi_n \wedge \neg\theta_1 \wedge \cdots \wedge \neg\theta_m$. Similarly, a *program* Π is a set of rules. And its universal closure is the conjunction of the universal closures of its rules.

Definition 15 (Equilibrium Model)**.** *Given a signature σ, a σ-interpretation $\mathcal{I}$ is an* equilibrium model *of a (set of) σ-sentence(s) ϕ if $(\mathcal{I}, \mathcal{I}, H) \models_\sigma \phi$ and for all $\mathcal{I}' \subsetneq \mathcal{I}$: $(\mathcal{I}', \mathcal{I}, H) \not\models_\sigma \phi$.*

It is well-known that equilibrium semantics of programs and answer set semantics of programs as defined in Section 2.1 align. Formally:

Proposition 16 ([PV06], Proposition 25)**.** *Let Π be a program. Then for every σ-interpretation $\mathcal{I}$ it holds that $\mathcal{I}$ is an equilibrium model of the universal closure of Π with respect to the classical signature $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ iff it is an answer set of Π over the predicates in $\mathcal{P}$, variables in $\mathcal{X}$ and domain $\mathcal{D}$.*

2.3.2 ASP($\mathcal{AC}$): ASP with Algebraic Constraints

We start by introducing first-order Weighted HT Logic. Intuitively it generalizes first-order HT Logic by replacing disjunctive connectives $(\vee, \exists)$ by additive ones $(+, \Sigma)$, conjunctive ones $(\wedge, \forall)$ by multiplicative ones $(*, \Pi)$ and accordingly the neutral elements $\bot, \top$ by "zero" and "one".

Definition 17 (Syntax)**.** *For a signature $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$, the* weighted σ*-formulas over the semiring $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ are of the form*

$$\alpha ::= k \mid x \mid \phi \mid \alpha \rightarrow_\mathcal{R} \alpha \mid \alpha + \alpha \mid \alpha * \alpha \mid -\alpha \mid \alpha^{-\mathbf{1}} \mid \Sigma y \alpha \mid \Pi y \alpha,$$

where $k \in R$, $x, y \in \mathcal{X}$ s.t. $\mathfrak{r}(s(x)) \subseteq R$ (i.e., x takes only values from R) and ϕ is a σ-formula. The use of $-$ and $^{-\mathbf{1}}$ require that $\oplus$ and $\otimes$ are invertible, the use of Πy requires that $\otimes$ is commutative. We define $\neg_\mathcal{R}\alpha = \alpha \rightarrow_\mathcal{R} e_\oplus$. A weighted σ*-sentence is a variable-free weighted σ-formula.*

Example 5. *Let $\sigma = \langle \mathbb{Q}, \{p\}, \{X\}, \{S\}, \{S \mapsto \mathbb{Q}\} \rangle$ and $s(X) = S$; thus, X ranges over the rational numbers. Then $\Sigma X p(X) * X$ is a weighted σ-sentence over the semirings $\mathbb{Q}, \mathbb{R}$ but not over $\mathbb{N}$.*

Definition 18 (Semantics)**.** *Let $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ be a signature. The semantics of a weighted σ-sentence over semiring $\mathcal{R}$ w.r.t. $\mathcal{I}_w = (\mathcal{I}^H, \mathcal{I}^T, w)$ is inductively defined as follows:*

$$[\![k]\!]^\sigma_\mathcal{R}(\mathcal{I}_w) = k, \;\; \text{for } k \in R$$

$$
\begin{aligned}
[\![-\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= -([\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w)) \\
[\![\alpha^{-1}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= ([\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w))^{-1} \\
[\![\phi]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= \begin{cases} e_\otimes, & \text{if } \mathcal{I}_w \models_\sigma \phi, \\ e_\oplus, & \text{otherwise.} \end{cases} \text{, for } \sigma\text{-formulas } \phi \\
[\![\alpha + \beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= [\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) \oplus [\![\beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) \\
[\![\alpha * \beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= [\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) \otimes [\![\beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) \\
[\![\alpha \to_{\mathcal{R}} \beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= \begin{cases} e_\otimes, & \text{if } [\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_{w'}) = e_\oplus \text{ or } [\![\beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_{w'}) \neq e_\oplus \text{ for all } w' \geq w, \\ e_\oplus, & \text{otherwise.} \end{cases} \\
[\![\Sigma x \alpha(x)]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= \begin{cases} \bigoplus_{\xi \in \mathrm{supp}_\oplus(\alpha(x), \mathcal{I}_w)} [\![\alpha(\xi)]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w), & \text{if } \mathrm{supp}_\oplus(\alpha(x), \mathcal{I}_w) \text{ is finite,} \\ \text{undefined}, & \text{otherwise.} \end{cases} \\
[\![\Pi x \alpha(x)]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= \begin{cases} \bigotimes_{\xi \in \mathrm{supp}_\otimes(\alpha(x), \mathcal{I}_w)} [\![\alpha(\xi)]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w), & \text{if } \mathrm{supp}_\otimes(\alpha(x), \mathcal{I}_w) \text{ is finite,} \\ e_\oplus, & \text{if } \mathfrak{r}(s(x)) \setminus \mathrm{supp}_\oplus(\alpha(x), \mathcal{I}_w) \neq \emptyset, \\ \text{undefined}, & \text{otherwise.} \end{cases}
\end{aligned}
$$

For the undefined value $e_\oplus^{-1}$ we use $e_\oplus$; here, $\mathrm{supp}_\odot(\alpha(x), \mathcal{I}_w)$ is the support *of $\alpha(x)$ w.r.t. $\mathcal{I}_w$ and $\odot \in \{\oplus, \otimes\}$, defined as*

$$\mathrm{supp}_\odot(\alpha(x), \mathcal{I}_w) \;=\; \{\xi \in \mathfrak{r}(s(x)) \mid [\![\alpha(\xi)]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) \neq e_\odot\},$$

i.e., the elements ξ in the range of x with a non-neutral value $\alpha(\xi)$ w.r.t. $\odot$.

Weighted HT Logic is a generalization of HT Logic in the following sense:

Proposition 19 (Generalization)**.** *Let ϕ be a σ-sentence and $\mathcal{I}_w$ be a pointed σ-HT-interpretation. Then, for the weighted σ-sentence α over the Boolean semiring $\mathbb{B}$, obtained from ϕ by replacing $\bot, \vee, \wedge, \to, \exists, \forall$ by $0, +, *, \to_{\mathbb{B}}, \Sigma, \Pi$, respectively, we have*

$$[\![\alpha]\!]^\sigma_{\mathbb{B}}(\mathcal{I}_w) = 1 \text{ iff } \mathcal{I}_w \models_\sigma \phi.$$

Proof. The claim can be easily verified by comparing the weighted semantics for $\mathcal{R} = \mathbb{B}$ and the unweighted semantics. We consider the case $\phi = \psi \to \theta$ in more detail. Then $\alpha = \beta \to_{\mathbb{B}} \theta$, where β and γ correspond to the rewritten versions of ψ and θ.

$$
\begin{aligned}
\mathcal{I}_w \models_\sigma \phi &\iff \mathcal{I}_{w'} \not\models_\sigma \psi \text{ or } \mathcal{I}_{w'} \models_\sigma \theta \text{ for } w' \geq w \\
&\iff [\![\beta]\!]^\sigma_{\mathbb{B}}(\mathcal{I}_w) \neq 1 \text{ or } [\![\gamma]\!]^\sigma_{\mathbb{B}}(\mathcal{I}_w) = 1 \text{ for } w' \geq w \\
&\iff [\![\beta]\!]^\sigma_{\mathbb{B}}(\mathcal{I}_w) = 0 \text{ or } [\![\gamma]\!]^\sigma_{\mathbb{B}}(\mathcal{I}_w) \neq 0 \text{ for } w' \geq w \\
&\iff [\![\alpha]\!]^\sigma_{\mathbb{B}}(\mathcal{I}_w) = 1
\end{aligned}
$$

□

The proof of the equivalence of $\to$ and $\to_{\mathcal{R}}$ works for arbitrary semirings $\mathcal{R}$: for σ-formulas ϕ, ψ the weighted formulas $\phi \to \psi$ and $\phi \to_{\mathcal{R}} \psi$ are equivalent. Thus, we can drop $\mathcal{R}$ from $\to_{\mathcal{R}}$.

Apart from being an HT Logic, the main difference between ours and the Weighted Logic introduced in [DG07] is that we allow for the additional connectives $-, {}^{-\mathbf{1}}$, and $\rightarrow$ and that ours is first-order over infinite domains instead of second-order over finite words. Another smaller difference is that we allow also unweighted formulas as subformulas of weighted formulas.

Defining a reasonable semantics for the case of infinite support seems challenging in general. For example, $\mathbb{Q}$ is neither closed under taking the limit of sequences, nor does every infinite sum of numbers converge, even in $\mathbb{R}$. For ω-continuous semirings such as $\mathbb{N}_\infty$, where both conditions above are satisfied, a definition would be possible. We omit this but refer the interested reader to [GKT07], where this is done in a similar context.

Intuitively, weighted formulas specify calculations over semirings depending on the truth of formulas. The quantifier Σ allows us to aggregate the values of weighted formulas for all variable assignments using $\oplus$ as the aggregate function.

Example 6 (cont.). *The semantics of $\Sigma X\ p(X) * X$ over $\mathcal{R}_{\max,+}$ is the maximum value x s.t. $p(x)$ holds. As $[\![p(x) * x]\!]^\sigma_{\mathcal{R}_{\max,+}}(\mathcal{I}_w) \neq e_\oplus = -\infty$ iff $p(x) \in \mathcal{I}^w$, we see that for finite $\mathcal{I}^w$*

$$\begin{aligned}[\![\Sigma X\ p(X) * X]\!]^\sigma_{\mathcal{R}_{\max,+}}(\mathcal{I}_w) &= \max\{[\![p(x) * x]\!]^\sigma_{\mathcal{R}_{\max,+}}(\mathcal{I}_w) \mid p(x) \in \mathcal{I}^w, x \in \mathbb{N}\} \\ &= \max\{0 + x \mid p(x) \in \mathcal{I}^w, x \in \mathbb{N}\} = \max\{x \mid p(x) \in \mathcal{I}^w\}.\end{aligned}$$

The semantic of weighted formulas is multi-valued in general. In order to return to the boolean semantics for programs, we define algebraic constraints, which are (in)equations between a semiring value and a weighted formula.

Definition 20 (Algebraic Constraints). *Let $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ be a signature. An* algebraic constraint *is an expression $k \sim_\mathcal{R} \alpha$ or $x \sim_\mathcal{R} \alpha$, where*

- *$(\mathcal{R}, >)$ is an ordered semiring, i.e., $\mathcal{R}$ is a semiring and $>$ is a strict total order on R,*
- *α is a weighted σ-formula over $\mathcal{R}$,*
- *$k \in R$ or $x \in \mathcal{X}$ such that $\mathfrak{r}(s(x)) \subseteq R$, and*
- *$\sim$ is a comparison operator from $\{>, \geq, =, \leq, <, \not>, \not\geq, \neq, \not\leq, \not<\}$.*

A sentence $k \sim_\mathcal{R} \alpha$ is satisfied w.r.t. $\mathcal{I}_w$, denoted $\mathcal{I}_w \models_\sigma k \sim_\mathcal{R} \alpha$, if

$$k \sim [\![\alpha]\!]^\sigma_\mathcal{R}(\mathcal{I}_{w'}) \text{ for all } w' \geq w.$$

The semantics for $x \sim_\mathcal{R} \alpha$ where the semiring value k is replaced by a variable x is left out intentionally since we only cover the semantics for sentences.

The syntax of σ-formulas in Section 2.3.1 is extended to include algebraic constraints in (2.18) as a further case. The definitions of satisfaction (Defn. 14) and equilibrium model (Defn. 15) are amended in the obvious way. However, as the semantics of weighted formulas is undefined for infinite supports, there are two variants of interpreting the condition $\mathcal{I}' \subsetneq \mathcal{I} : (\mathcal{I}', \mathcal{I}, H) \not\models_\sigma \phi$ in defining equilibrium models. If we adopt that $\not\models_\sigma$ holds when the semantics is undefined, we end up with *weak* equilibrium models, otherwise with *strong* equilibrium models.

To verify that the semantics of algebraic constraints is in line with the intuition of HT logic, we show that the persistence property is maintained for sentences that include algebraic constraints.

Proposition 21 (Persistence)**.** *For any σ-sentence ϕ and σ-HT-interpretation $(\mathcal{I}^H, \mathcal{I}^T)$, it holds that $\mathcal{I}_H \models_\sigma \phi$ implies $\mathcal{I}_T \models_\sigma \phi$.*

Proof. It is know that the proposition holds for formulas ϕ without algebraic constraints [PV06]. We can use the same proof by structural induction, given that we can prove that the claim holds for the additional base case $\phi = k \sim_\mathcal{R} \alpha$.

In this case however, the definition of satisfaction tells us that

$$\mathcal{I}_w \models_\sigma k \sim_\mathcal{R} \alpha \iff k \sim [\![\alpha]\!]^\sigma_\mathcal{R}(\mathcal{I}_{w'}), \text{ for all } w' \geq w.$$

So, since $T \geq H$ from $\mathcal{I}_H \models_\sigma k \sim_\mathcal{R} \alpha$ follows $\mathcal{I}_T \models_\sigma k \sim_\mathcal{R} \alpha$. □

Note that this also entails that one can not support atoms by using constraints $\bot \leftarrow k \sim_\mathcal{R} \alpha$. E.g. $\bot \leftarrow 0 =_\mathbb{N} a$ does not have $\{a\}$ as a stable model.

Having established that the semantics behaves as desired, we formally define programs that can contain algebraic constraints in terms of a fragment of the logic over *semiring signatures*.

Definition 22 (Semiring Signature)**.** *A signature $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ is a* semiring signature *for semirings $\mathcal{R}_1, \ldots, \mathcal{R}_n$, where $\mathcal{R}_i = \langle R_i, \oplus_i, \otimes_i, e_{\oplus_i}, e_{\otimes_i} \rangle$, $i = 1, \ldots, n$, if*

1. *$\mathcal{S}$ is a superset of $2^{\{1,\ldots,n\}}$,*
2. *$\mathcal{D}$ contains R_i, for all $i = 1, \ldots, n$, and*
3. *$\mathfrak{r} : \mathcal{S} \to 2^\mathcal{D}$ maps $\{i_1, \ldots, i_m\}$ to $\bigcap_{j=1}^m R_{i_j}$.*

Intuitively, if a variable x has sort $\{i_1, \ldots, i_m\}$, then we only want to quantify over those domain-values that are in every semiring $\mathcal{R}_{i_1}$ to $\mathcal{R}_{i_m}$. Imagine for example that a variable x is used as a placeholder for a semiring value in two algebraic constraints, one over $\mathbb{N}$ and one over $\mathbb{Q}$. Then it only makes sense to quantify over domain-values that are contained in $\mathbb{N}$.

Definition 23 ($\mathcal{AC}$-Rules, $\mathcal{AC}$-Programs)**.** *Let $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ be a semiring signature for $\mathcal{R}_1, \ldots, \mathcal{R}_n$. Then an $\mathcal{AC}$-programs is a set of $\mathcal{AC}$-rules r of the form*

$$r = H(r) \leftarrow B(r) = \phi \leftarrow \psi_1, \ldots, \psi_n, \neg\theta_1, \ldots, \neg\theta_m, \tag{2.19}$$

where each ϕ, ψ_i and θ_j is either a σ-atom or an algebraic constraint over $\mathcal{R}_i$ for some $i = 1, \ldots, n$, in which no quantifiers or nested constraints occur. Furthermore, we require for each variable x occurring in r that $i \in s(x)$ iff x occurs in place of a value from the semiring $\mathcal{R}_i$.

Example 7 (Rules)**.** *The following are examples of $\mathcal{AC}$-rules:*

$$loc\_sum(Y) \leftarrow Y =_{\mathbb{Q}} ind(I) * loc\_weight(I, W) * W \tag{2.20}$$
$$glob\_sum(Y) \leftarrow glob\_weight(W), Y =_{\mathbb{Q}} ind(I) * W \tag{2.21}$$

Note that in $\mathcal{AC}$-rules quantifiers occur neither in weighted nor in unweighted formulas. Variables are quantified implicitly, depending on their scope defined as follows.

Definition 24 (Local & Global)**.** *A variable x that occurs in an $\mathcal{AC}$-rule r is* local*, if it occurs in r only in weighted formulas, and* global *otherwise. A atom, rule or program is* locally *(resp.* globally*)* ground*, if it has no local (resp. global) variables.*

Example 8 (cont.)**.** *In the previous example Y and I are respectively global and local in both rules, whereas W is local in rule (2.20) and global in rule (2.21).*

We then quantify global variables universally and local variables "existentially" (i.e. using Σ).

Definition 25 (Program and Rule Semantics)**.** *Let r be an $\mathcal{AC}$-rule of the form (2.19) that contains global variables $x_1, \ldots, x_k$. Its semantics is that of the σ-formula*

$$\forall x_1, \ldots, x_k \; (B_\wedge(r) \to \phi)^{\Sigma}, \qquad \text{where } B_\wedge(r) = \psi_1 \wedge \cdots \wedge \psi_n \wedge \neg\theta_1 \wedge \cdots \wedge \neg\theta_m$$

and $(\cdot)^{\Sigma}$ replaces every weighted formula α with local variables $y_1, \ldots, y_l$ by $\Sigma y_1, \ldots, y_l \; \alpha$.

Example 9 (cont.)**.** *Consequently, the $\mathcal{AC}$-rules from above correspond to the formulas*

$$\forall Y \; (Y =_{\mathbb{Q}} \Sigma I \; \Sigma W \; ind(I) * loc\_weight(I, W) * W) \to loc\_sum(Y)$$
$$\forall Y \; \forall W \; glob\_weight(W) \wedge (Y =_{\mathbb{Q}} \Sigma I \; ind(I) * W) \to glob\_sum(Y).$$

We see that rule (2.20) calculates the sum over all indices $\{i \mid ind(i)\}$ weighted locally with w when $loc\_weight(i, w)$ holds. Rule (2.21) calculates the sum over all indices $\{i \mid ind(i)\}$ where all of them are weighted with the same weight w when $glob\_weight(w)$ holds.

Note that we strongly restricted the weighted formulas that are allowed in $\mathcal{AC}$-programs. The quantifier Π and nested algebraic constraints are unavailable and Σ quantifiers can only occur as a prefix. Removing these restrictions would lead to a much higher complexity. Already constraint evaluation would be PSPACE-hard for any non-trivial semiring. In addition, our choice allows us to keep the syntax of $\mathcal{AC}$-programs closer to the one of other programs with constraints.

In the sequel, we drop $\mathcal{AC}$ from $\mathcal{AC}$-rules and $\mathcal{AC}$-programs if no ambiguity arises.

Example 10 (Metro Connections)**.** *We reconsider the metro connections example. We can use the following weighted formula α_{enter} to simplify the program Π_m:*

$$\neg\neg depart(M,S) * \neg exit(M,S) * ((depart(M,S) * \neg exit(M,S)) \rightarrow enter(M,S)).$$

See 2.3.4 for a general explanation of formulas of this form.

We use the rule

$$1 =_{\mathbb{N}} \alpha_{enter} \leftarrow reach(S), not\, goal(S) \tag{2.22}$$

to replace the rules

$$\{enter(M,S)\} \leftarrow reach(S), depart(M,S) \tag{2.23}$$
$$\leftarrow enter(M,S), exit(M,S) \tag{2.24}$$
$$\leftarrow enter(M1,S), enter(M2,S), M1 \neq M2 \tag{2.25}$$

from the previous example. The idea behind Rule 2.22 is the following: if we reach a station S that is not the goal station, then we want to enter another metro. However, we only want to enter metros that depart at station S and that we did not exit at S. This covers the first two replaced rules. Moreover, since we assert that the value of the weighted formula is equal to 1, there is exactly one metro that we enter at station S. Thus, also the satisfaction of the last rule follows, making it redundant.

As a side note, in the syntax that is currently accepted by the clingo solver [Geb+14] Rule 2.22 would take the form

$$1 = \#count\{M : enter(M,S) : depart(M,S), not\, exit(M,S)\} \leftarrow reach(S), not\, goal(S).$$

Apart from this, we can model that only metro connections whose cost is within our available budget are possible. Without quantitative constraints this is complicated. However, with algebraic constraints, given $budget(b)$ and $cost(m,c)$ as input predicates denoting a budget of b and cost c for taking the metro m, we can use the following rule:

$$\leftarrow budget(B), B <_{\mathbb{Q}} enter(M,S) * cost(M,C) * C.$$

Here, the weighted formula $enter(M,S) * cost(M,C) * C$ sums up all the costs of metros that we enter. Note that if we enter a metro twice at different stations, which might lead

to a shorter connection, this formula obliges us to buy two separate tickets. If this is not desired, we can use the following rules instead:

$$\begin{aligned} use(M) &\leftarrow enter(M, S) \\ &\leftarrow budget(B), B <_{\mathbb{Q}} use(M) * cost(M, C) * C \end{aligned}$$

The above example shows that algebraic constraints indeed facilitate the specification of complicated programs, especially, when we want to reason with constraints on numeric quantities. In the following, we consider a more advanced example use case of $\mathcal{AC}$-programs in the context of provenance.

Example Application: Provenance

Green, Karvounarakis, and Tannen [GKT07] introduced a semiring-based semantics that is capable of expressing bag semantics, why-provenance and more. For positive logic programs, their semantics over a semiring $(R, \oplus, \otimes, e_\oplus, e_\otimes)$ is as follows: the label of a query result $q(\overline{x})$ is the sum (using $\oplus$) of the labels of derivation trees for $q(\overline{x})$, where the label of a derivation tree is the product (using $\otimes$) of the labels of the leaf nodes (i.e. extensional atoms). As the number of derivation trees may be countably infinite, Green et al. used ω-continuous semirings such as $\mathbb{N}_\infty$ that allow to have countable sums.

Example 11 (Bag Semantics)**.** *For ease of exposition, consider the propositional program*

$$r_1\colon b \leftarrow e_1, e_2 \qquad r_2\colon b \leftarrow e_1 \qquad r_3\colon c \leftarrow e_2, b \qquad r_4\colon c \leftarrow c, c$$

over $\mathbb{N}_\infty$, which corresponds to bag semantics (cf. [GKT07]) and the extensional database (edb) $\{(e_1, 2), (e_2, 0)\}$. The label of b under bag semantics is $2{+}0{\cdot}2 = 2$. Here 2 corresponds to the derivation from $r_2, (e_1, 2)$ and $0 \cdot 2$ to the derivation from $r_1, (e_1, 2), (e_2, 0)$. The label of c is 0 as it can only be derived using e_2.

We can model the semiring semantics in our formalism, by allowing operations over countable supports $\mathrm{supp}_\odot(\alpha(x), \mathcal{I}_w)$ for ω-continuous semirings. Over $\mathbb{N}_\infty$ they always have the value ∞.

Example 12 (cont.)**.** *The following $\mathcal{AC}$-program calculates the provenance semantics over $\mathbb{N}_\infty$ for the above positive logic program, depending on the edb:*

$$1 =_{\mathbb{B}} p(b, 1, 2, X) * d(b, 2) \leftarrow p(e_1, 1, X_1), p(e_2, 1, X_2), X =_{\mathbb{N}_\infty} X_1 + X_2 \tag{2.26}$$

$$1 =_{\mathbb{B}} p(b, 2, 1, X) * d(b, 1) \leftarrow p(e_1, 1, X) \tag{2.27}$$

$$1 =_{\mathbb{B}} p(c, 3, V, X) * d(c, V) \leftarrow p(e_2, 1, X_1), p(b, V_1, X_2), V =_{\mathbb{N}_\infty} V_1 + 1, X =_{\mathbb{N}_\infty} X_1 + X_2 \tag{2.28}$$

$$1 =_{\mathbb{B}} p(c, 4, V, X) * d(c, V) \leftarrow p(c, V_1, X_1), p(c, V_2, X_2), V =_{\mathbb{N}_\infty} V_1 + 1, X =_{\mathbb{N}_\infty} X_1 + X_2 \tag{2.29}$$

$$1 =_{\mathbb{B}} p(A, V, X) \leftarrow d(A, V), X =_{\mathbb{N}_\infty} p(A, I, V, X^*) * X^* \tag{2.30}$$

$$1 =_{\mathbb{B}} f(A, X) \leftarrow d(A, V), X =_{\mathbb{N}_\infty} p(A, V^*, X^*) * X^* \tag{2.31}$$

Here $p(A, V, X)$ represents that X is the sum of all labels of derivation trees for A having exactly V many leaf nodes. We obtain this value first for all derivation trees that apply rule r_i last, in $p(A, i, V, X)$, and sum them up in rule (2.30). Similarly the final provenance value is obtained as the sum over the provenance values for each number of leaf nodes V^* in rule (2.31); $d(A, V)$ says that there is a derivation tree of A using V leaf nodes and ensures safety (see next section).

We can apply this strategy in general: even for a non-ground positive logic program we can give an $\mathcal{AC}$-program that computes the provenance semantics in a similar fashion as in the example above.

Theorem 26 (Provenance Encoding)**.** *Given a positive datalog program Π there is an $\mathcal{AC}$-program $T(\Pi)$ that computes the provenance semantics over the ω-continuous semiring $\mathcal{R}$ in the following sense. Let D be an edb and $r(\overline{x})$ a query result of $D \cup \Pi$ with semiring provenance v. Then the unique equilibrium model $\mathcal{I}$ of $T(\Pi) \cup \{p_e(\overline{x}, v, 1) \leftarrow\mid (e(\overline{x}), v) \in D\}$ contains $p_r(\overline{x}, v')$ iff $v' = v$. Where the predicates of the form $p_s(\overline{x}, v, \overline{y})$ correspond to original predicates s, with semiring label v and potential auxiliary parameters $\overline{y}$.*

Proof (Sketch, see Appendix A.1 for the full proof). Intuitively, the idea is that we can define predicates that compute the provenance along hypothetical derivation trees, by storing the rules that were used last and the number of extensional atoms that were used. □

This extended example of how provenance can be modeled using algebraic constraints shows that they are quite powerful and flexible, when it comes to quantitative computations.

2.3.3 Language Aspects

In this subsection, we consider different aspects of the language of $\mathcal{AC}$-programs. Namely, we consider safety and program equivalence.

Domain Independence and Safety

We need to restrict ourselves to programs that are well behaved, i.e. independent of the domain they are evaluated over.

Example 13. *Consider the weighted formula $\alpha = \Sigma x\ \neg q(x)$, which counts the elements d in the domain s.t. $q(d)$ does not hold. It is easy to see that if we consider the semantics using the same interpretation but over different domains (or rather signatures) it can vary.*

We are interested in formulas that do not exhibit this kind of behavior, formalized as:

Definition 27 (Domain Independence)**.** *A sentence ϕ (resp. weighted sentence α over semiring $\mathcal{R}$) is* domain independent*, if for every two semiring signatures $\sigma_i = \langle \mathcal{D}_i, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r}_i \rangle (i = 1, 2)$ s.t. ϕ is a σ_i-formula (resp. α is a weighted σ_i-formula) for $i = 1, 2$ and every $\mathcal{I}_w = (\mathcal{I}^H, \mathcal{I}^T, w)$ that is a pointed σ_i-HT-interpretation for $i = 1, 2$ it holds that*

$$\mathcal{I}_w \models_{\sigma_1} \phi \text{ iff } \mathcal{I}_w \models_{\sigma_2} \phi \quad (\text{resp. } [\![\alpha]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) = [\![\alpha]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w)).$$

Note that we restrict ourselves to semiring signatures, which implies that all semiring values are elements of the domain.

We restrict ourselves to a fragment of weighted formulas. Intuitively, we need to ensure that every variable X in $\alpha(\overline{X})$ is bound by a positive occurrence of a predicate $p(X)$.

Definition 28 (Syntactic Domain Independence)**.** *A weighted formula $\alpha(\overline{X})$ over a semiring $\mathcal{R}$ is* syntactically domain independent *w.r.t. $\overline{X}$, if it is constructible following*

$$\begin{aligned}\phi(\overline{X}) ::= &\bot \mid p(\overline{X}) \mid \neg\neg\phi(\overline{X}) \mid \phi(\overline{X}) \vee \phi(\overline{X}) \mid \phi(\overline{Y}) \wedge \phi(\overline{Z}) \mid \phi(\overline{X}) \wedge \psi(\overline{X'}),\\ \alpha(\overline{X}) ::= &k \mid \phi(\overline{X}) \mid \neg\neg\alpha(\overline{X}) \mid \alpha(\overline{X}) + \alpha(\overline{X}) \mid \alpha(\overline{Y}) * \alpha(\overline{Z}) \mid \alpha(\overline{X}) * \beta(\overline{X'})\\ &\mid -\alpha(\overline{X}) \mid \alpha^{\mathbf{-1}}(\overline{X}),\end{aligned}$$

where $k \in R$, $p(\overline{X})$ is an atom, $\psi(\overline{X'})$ ($\beta(\overline{X'})$) is any (weighted) formula, $\overline{X'} \subseteq \overline{X}$ and $\overline{Y} \cup \overline{Z} = \overline{X}$.

Intuitively, X can only occur in α, if it occurs in a non-negated or double negated predicate in every branch of a sum or disjunction. To allow the usage of arbitrary formulas $\psi(\overline{X'})$ (resp. $\beta(\overline{X'})$) in the case $\alpha(\overline{X}) * \beta(\overline{X'})$ (resp. $\phi(\overline{X}) \wedge \psi(\overline{X'})$) the variables in $\overline{X'}$ already need to be "guarded" by $\alpha(\overline{X})$ (resp. $\phi(\overline{X})$).

Example 14 (cont.)**.** *While $\neg q(Y)$ from Example 13 is not syntactically domain independent w.r.t. Y, the formula $p(Y) * \neg q(Y)$, which counts the number d s.t. $p(d)$ holds but not $q(d)$, is. It can be constructed using $\alpha(\overline{X}) * \beta(\overline{X'})$.*

Our syntactic criterion guarantees domain independence.

Theorem 29 (Formula Domain Independence)**.** *If a formula $\alpha(\overline{X})$ over semiring $\mathcal{R}$ is syntactically domain independent w.r.t. $\overline{X}$, then $\alpha^{\Sigma} = \Sigma\overline{X}\ \alpha(\overline{X})$ is domain independent.*

Proof (Sketch, see Appendix A.2 for the full proof). Invariance of $\text{supp}_{\oplus}(\alpha(x), \mathcal{I}_w)$ w.r.t. σ_i (or rather $\mathcal{D}_i$) is shown by structural induction. We show the invariance for one interesting case, namely $\alpha = \alpha_1(x) * \alpha_2$. Note that:

$$\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_1(\xi) * \alpha_2]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\} \subseteq \{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_1(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}$$

Therefore, we obtain

$$\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_1(\xi) * \alpha_2]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}$$

$$=\{\xi \in \{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_1(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\} \mid [\![\alpha_1(\xi) * \alpha_2]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\}$$

Next, we use the induction hypothesis on $\alpha_1(x)$ to obtain

$$\begin{aligned}&=\{\xi \in \{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\alpha_1(\xi)]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\} \mid [\![\alpha_1(\xi) * \alpha_2]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\}\\ &=\{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\alpha_1(\xi) * \alpha_2]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\}.\end{aligned}$$

As the semantics of variable-free formulas is domain independent the claim follows. □

Safety of programs is defined as follows.

Definition 30 (Safety)**.** *A program* Π *is safe, if each rule* $r \in \Pi$ *of form equation* (2.19) *is safe, i.e. fulfills that*

(i) *every weighted formula in* r *is syntactically domain independent w.r.t. its local variables;*

(ii) *for every global variable* X *there exists some* β_i *s.t. (1)* β_i *is an atom and* X *occurs in it, or (2)* β_i *is* $X =_{\mathcal{R}} \beta_i'$ *and* X *does not occur in any weighted formula in the body of* r*.*

The restriction in (ii) that X does not reoccur is necessary to prohibit $p(X) \leftarrow X =_{\mathcal{R}} Y, Y =_{\mathcal{R}} X$. It could however be replaced by a more sophisticated acyclicity condition.

Example 15 (Safety)**.** *The rules (2.20) and (2.21) are safe. Without the predicate* p *the rule*

$$c(X) \leftarrow X =_{\mathbb{N}} p(X) * \neg q(X)$$

would not be safe.

Theorem 31 (Program Domain Independence)**.** *Safe programs are domain independent.*

Proof. Let $\sigma_i = \langle \mathcal{D}_i, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r}_i \rangle, i = 1, 2$ be semiring signatures s.t. Π is a σ_i-formula for $i = 1, 2$ and let $\mathcal{I}_w = (\mathcal{I}^H, \mathcal{I}^T, w)$ be a pointed σ_i-HT-interpretation for $i = 1, 2$.

Let $r \in \Pi$. If r does not contain global variables, the claim is evident. Otherwise assume $r = \forall x_1, \ldots, x_n\, \alpha(x_1, \ldots, x_n)$. When $\xi_j \in \mathfrak{r}_1(s(x_j)) \cap \mathfrak{r}_2(s(x_j))$ $(j = 1, \ldots, n)$, the semantics of $\alpha(\xi_1, \ldots, \xi_n)$ does not depend on σ_i. Suppose that $\xi_j \in \mathfrak{r}_1(s(x_j)) \setminus \mathfrak{r}_2(s(x_j))$. Then x_j cannot occur in place of a semiring value as for semiring signatures, we have $\mathfrak{r}_1(s(x_j)) = \bigcap_{j=1}^m R_{k_j} = \mathfrak{r}_2(s(x_j))$. Therefore, x_j has to satisfy item (ii.1) of safety, implying that some atom β_k in the body of r is not satisfied by $\mathcal{I}_w$ and hence $\mathcal{I}_w \not\models_{\sigma_i} \alpha(\xi_1, \ldots, \xi_n)$ for $i = 1, 2$. □

Not every domain independent program is safe. E.g. $p(X) \leftarrow \top =_{\mathbb{B}} q(X)$ is not safe but is equivalent to the safe rule $p(X) \leftarrow q(X)$ since X is a global variable and we can only derive $p(x)$ when $[\![q(x)]\!]_{\mathbb{B}} = 1$, i.e. when $q(x)$ holds. Domain independence is undecidable but safety is sufficient, allows for complex rules like equation (2.20), equation (2.21), and those in Example 12, and is easily checked.

In the rest of the section, we restrict ourselves to domain independent programs and can therefore remove the annotation σ from $\models_\sigma$ and $[\![\cdot]\!]^\sigma_{\mathcal{R}}$ and use $\models$ and $[\![\cdot]\!]_{\mathcal{R}}$ instead. Accordingly, we do not need to specify the signature for $\mathcal{AC}$-programs Π anymore, as any semiring signature σ s.t. Π is an $\mathcal{AC}$-program over σ suffices.

Program Equivalence

An additional benefit of HT-semantics is that we are able to characterize strong program equivalence as equivalence in the logic of HT.

Definition 32 (Strong Equivalence)**.** *Programs* Π_1 *and* Π_2 *are strongly equivalent, denoted by* $\Pi_1 \equiv_s \Pi_2$*, if for every program* Π' *the equilibrium models of* $\Pi_1 \cup \Pi'$ *and* $\Pi_2 \cup \Pi'$ *coincide.*

Characterization results have already been proven for classical ASP programs with [PV08] or without variables [LPV01] and many more. As for classical ASP programs, the following characterization of strong equivalence holds:

Theorem 33 (Strong Equivalence)**.** *For any* Π_1, Π_2 *programs,* $\Pi_1 \equiv_s \Pi_2$ *iff* Π_1 *has the same HT-models, i.e. satisfying pointed HT-interpretations, as* Π_2*.*

Proof (Sketch, see Appendix A.2 for the full proof). The direction $\Leftarrow$ is clear. For $\Rightarrow$ we can generalize the proof in [LPV01], by constructing Π', which asserts a subset of the interpretation $\mathcal{I}^T$ that is ensured to be stable ($\mathcal{I}^H$), and a subset that if partly present is ensured to be fully present ($\mathcal{I}^T \setminus \mathcal{I}^H$).

Let Π_1 and Π_2 have different HT-models. W.l.o.g. there must be at least one HT-interpretation $(\mathcal{I}^H, \mathcal{I}^T)$ that is an HT-model of Π_1 but not of Π_2. As in [LPV01] we simply define

$$\Pi' = \{p(\overline{x}) \leftarrow \mid p(\overline{x}) \in \mathcal{I}^H\} \cup \{p(\overline{x}) \leftarrow q(\overline{y}) \mid p(\overline{x}), q(\overline{y}) \in \mathcal{I}^T \setminus \mathcal{I}^H\}$$

Then $\mathcal{I}^T$ is an equilibrium model of $\Pi_2 \cup \Pi'$, but not of $\Pi_1 \cup \Pi'$ and therefore Π_1 and Π_2 are not strongly equivalent. □

Note that since $\mathcal{I}^H$ may be infinite, this may result in programs of infinite size. This can be circumvented if auxiliary predicates are allowed in Π'. For example, in [LPV07] the strong equivalence of arbitrary first-order formulas was considered and characterized as equivalence in HT Logic. The proof however uses the fact that the strong equivalence considered in their work is for any first-order sentence and not only for programs, which

Construct	ASP($\mathcal{AC}$)	Others
Nested Expressions	$1 =_{\mathbb{B}} \alpha \leftarrow 1 =_{\mathbb{B}} \beta$	$\alpha \leftarrow \beta$
Aggregates	$T \sim_{\mathbb{Q}} (p(X) + q(X)) * X$	$T \sim \text{sum}\{X : p(X), X : q(X)\}$
Choice	$k \leq^{c}_{\mathcal{R}} \neg\neg q(X,W) * (q(X,W) \rightarrow p(X)) * W$	$k \leq \{p(X) : q(X,W) = W\}$
Minimized Choice	$k \leq_{\mathcal{R}} \neg\neg q(X,W) * (q(X,W) \rightarrow p(X)) * W$	n/a
Value Guess	$k \leq_{\mathcal{R}} \text{val}(X) * X$	$k \leq$ val (CP+ASP)
Arithmetics	$X =_{\mathbb{Q}} Y * Z^{-1}$, $s \geq_{2^A} X + Y$	$X = Y \div Z$, $s \supseteq X \cup Y$

Table 2.1: Constructs expressible in ASP($\mathcal{AC}$) and how they are expressed in other formalisms.

are a syntactic fragment. A straight-forward way to reproduce their proof strategy in our setting seems not to be apparent.

Nevertheless, it is possible to prove the statement when programs are finite sets of rules in our setting, provided that auxiliary predicate symbols are available not occurring in the program (which trivially holds if we have infinitely many predicates of each arity in the underlying predicate signature $\mathcal{P}$). This is described in detail in Appendix A.3.

2.3.4 Relation to Similar Formalisms

We consider several constructs that we can express in ASP($\mathcal{AC}$) and relate them to constructs known from previous extensions of ASP; a summary is given in Table 2.1.

Nested Expressions The logic programs with arbitrary propositional formulas defined in [LTT99] are modeled simply using constraints over the Boolean semiring $\mathbb{B}$. As a special case, this shows the expressibility of disjunctive logic programming rules

$$a_1 \vee \cdots \vee a_n \leftarrow B(r)$$

using

$$1 =_{\mathbb{B}} a_1 + \ldots + a_n \leftarrow B(r).$$

Conditionals Cabalar, Fandinno, Schaub, and Wanko [Cab+20b] defined two semantics for conditionals $s = (s'|s'' : \phi)$, where s', s'' are terms and ϕ is a (quantifier-free) formula, named *vicious circle (vc)* and *definedness (df)*, respectively. Given an interpretation $(\mathcal{I}^H, \mathcal{I}^T)$,

$$vc_{\mathcal{I}_w}(s) = \begin{cases} s', & \text{if } \mathcal{I}_w \models_\sigma \phi, \\ s'', & \text{if } \mathcal{I}_w \models_\sigma \neg\phi, \\ \text{undefined}, & \text{otherwise.} \end{cases} \qquad df_{\mathcal{I}_w}(s) = \begin{cases} s', & \text{if } \mathcal{I}_w \models_\sigma \phi, \\ s'', & \text{otherwise.} \end{cases}$$

Syntax and semantics of weighted formulas could be readily extended to include these constructs. We present instead an alternative evaluation of conditionals as formulas $s' * \phi + s'' * \neg\phi$. Then

$$[\![s' * \phi + s'' * \neg\phi]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) = \begin{cases} s', & \text{if } \mathcal{I}_w \models_\sigma \phi, \\ s'', & \text{if } \mathcal{I}_w \models_\sigma \neg\phi, \\ e_\oplus, & \text{otherwise.} \end{cases} \tag{2.32}$$

The intuition here is that without a reason no effect should occur and thus none of the two values should be taken. That is, when neither ϕ nor $\neg\phi$ is satisfied, we end up with the neutral element $e_\oplus$. Therefore, if we do not have any information the conditional has no effect, instead of leading to undefinedness or s''. This combines, in a sense, the strengths of both *df* and *vc* as the following example shows.

Example 16. *Consider the following rules r_1 and r_2:*

$$r_1 = p \leftarrow \top = (\top \mid \bot : p) \vee \top \qquad\qquad r_2 = p \leftarrow \top = (\top \mid \top : p).$$

According to vc *resp.* df*, they are equivalent: under* vc*, both have no stable model while under* df *both have the stable model $\{p\}$. We may expect that r_1 has the stable model $\{p\}$ as the formula $s \vee \top$ is equivalent to $\top$ regardless of the value of s. Therefore, the value of p should not influence the truth of the body of r_1. On the other hand, the value of the conditional in r_2 influences the truth of the body of r_2 and it depends on p. Therefore, if $\{p\}$ were a stable model of r_2, we would arguably derive p using the truth value of p. Accordingly, we may expect that r_2 does not have a stable model. These expectations align with the semantics for r_1 and r_2 from (2.32) above.*

This evaluation combines the ideas behind the *vc* and the *df* principle: the value of a conditional is always defined, but the vicious circle of deriving p by the truth value of p is avoided.

Apart from that, we may express *vc* and *df* in our formalism. Since we do not capture arbitrary constraints as Cabalar et al. do, we assume instead that conditionals in weighted formulas are allowed and show that it is unnecessary to allow them explicitly. We start with *vc*.

Let $r(s) = H(r(s)) \leftarrow B(r(s))$ be some rule containing a conditional $s = (s'|s'' : \phi)$ which is supposed to be evaluated under *vc* semantics. This means that

$$\mathcal{I}_w \models r(s) \iff \begin{cases} \mathcal{I}_w \models r(s') & \text{if } \mathcal{I}_w \models \phi \\ \mathcal{I}_w \models r(s'') & \text{if } \mathcal{I}_w \models \neg\phi \\ \mathcal{I}_T \models r(s) & \text{otherwise.} \end{cases} .$$

Now if we simply replace s by $\phi * s' + \neg\phi * s''$ we get

$$\mathcal{I}_w \models r(\phi * s' + \neg\phi * s'') \iff \begin{cases} \mathcal{I}_w \models r(s') & \text{if } \mathcal{I}_w \models \phi \\ \mathcal{I}_w \models r(s'') & \text{if } \mathcal{I}_w \models \neg\phi \\ \mathcal{I}_T \models r(s) \text{ and } \mathcal{I}_w \models r(e_\oplus) & \text{otherwise.} \end{cases} .$$

This is obviously different, however if we use instead the rule

$$r'(s) = H(r(s)) \leftarrow B(r(s)), 1 =_{\mathbb{B}} \phi + \neg\phi$$

we obtain

$$\begin{aligned} & \mathcal{I}_w \models r'(\phi * s' + \neg\phi * s'') \\ \iff & \begin{cases} \mathcal{I}_w \models r'(s') & \text{if } \mathcal{I}_w \models \phi \\ \mathcal{I}_w \models r'(s'') & \text{if } \mathcal{I}_w \models \neg\phi \\ \mathcal{I}_T \models r'(s) \text{ and } \mathcal{I}_w \models H(r(e_\oplus)) \leftarrow B(r(e_\oplus)), 1 =_{\mathbb{B}} \phi + \neg\phi & \text{otherwise.} \end{cases} \\ \iff & \begin{cases} \mathcal{I}_w \models r(s') & \text{if } \mathcal{I}_w \models \phi \\ \mathcal{I}_w \models r(s'') & \text{if } \mathcal{I}_w \models \neg\phi \\ \mathcal{I}_T \models r(s) \text{ and } \mathcal{I}_w \models H(r(e_\oplus)) \leftarrow B(r(e_\oplus)), 1 =_{\mathbb{B}} 0 & \text{otherwise.} \end{cases} \\ \iff & \begin{cases} \mathcal{I}_w \models r(s') & \text{if } \mathcal{I}_w \models \phi \\ \mathcal{I}_w \models r(s'') & \text{if } \mathcal{I}_w \models \neg\phi \\ \mathcal{I}_T \models r(s) & \text{otherwise.} \end{cases} \end{aligned}$$

as desired.

In order to model *df*, we further need that the addition $\oplus$ of the semiring $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ is invertible, i.e. that we can use the connective $-$. Assume this is the case and let $s = (s' \mid s'' : \phi)$ be a conditional over the semiring $\mathcal{R}$ that we want to evaluate under *df*. Then its semantics is

$$df_{\mathcal{I}_w}(s) = \begin{cases} s', & \text{if } \mathcal{I}_w \models_\sigma \phi, \\ s'', & \text{otherwise.} \end{cases}$$

We can use the weighted formula $\phi * s' + (e_\otimes + -\phi) * s''$ and obtain

$$\begin{aligned} [\![\phi * s' + (e_\otimes + -\phi) * s'']\!]_{\mathcal{R}}(\mathcal{I}_w) &= [\![\phi * s']\!]_{\mathcal{R}}(\mathcal{I}_w) \oplus [\![(e_\otimes + -\phi) * s'']\!]_{\mathcal{R}}(\mathcal{I}_w) \\ &= \begin{cases} e_\otimes \otimes [\![s']\!]_{\mathcal{R}}(\mathcal{I}_w) \oplus (e_\otimes \oplus - e_\otimes) \otimes [\![s'']\!]_{\mathcal{R}}(\mathcal{I}_w) & \mathcal{I}_w \models \phi \\ e_\oplus \otimes [\![s']\!]_{\mathcal{R}}(\mathcal{I}_w) \oplus (e_\otimes \oplus - e_\oplus) \otimes [\![s'']\!]_{\mathcal{R}}(\mathcal{I}_w) & \text{otherwise} \end{cases} \end{aligned}$$

$$= \begin{cases} s' \oplus e_\oplus \otimes s'' & \mathcal{I}_w \models \phi \\ e_\otimes \otimes s'' & \text{otherwise} \end{cases}$$
$$= \begin{cases} s' & \mathcal{I}_w \models \phi \\ s'' & \text{otherwise} \end{cases}$$

Formally, this means that

Corollary 34. *Let $\mathcal{R}$ be a semiring and $r(s)$ be a rule that contains a conditional $s = (s'|s'' : \phi)$ in an algebraic constraint over $\mathcal{R}$. Then*

- *we can model $r(s)$ under* vc *semantics using the $\mathcal{AC}$-rule*

$$H(r(\phi * s' + \neg\phi * s'')) \leftarrow B(r(\phi * s' + \neg\phi * s'')), 1 =_{\mathbb{B}} \phi + \neg\phi.$$

- *if $\oplus$ is invertible, then we can model $r(s)$ under* df *semantics using the $\mathcal{AC}$-rule*

$$r(\phi * s' + (e_\otimes + -\phi) * s'').$$

Summarizing, the possibility to express *vc* and *df* as well as to define other semantics of conditionals exemplifies the power of FO-WHT Logic and $\mathcal{AC}$-programs.

Constraints in the head for guessing In many ASP extensions, constraints in rule heads and rule bodies behave differently; in heads, they are used as *choice constraints*. Consider the rule

$$1 = \#count\{M : \text{enter}(M,S) : \text{depart}(M,S), \text{not exit}(M,S)\} \leftarrow \text{reach}(S), \text{not goal}(S).$$

in lparse syntax from Example 10.

It specifies that, when we reach a station (reach(S)) that is not the goal station (not goal(S)), then we must enter exactly one metro at that station (enter(M,S)) which departs at that station (depart(M,S)) and which we did not exit at that station (exit(M,S)).

In order to express this constraint in the head in our semantics, we need to take care that it may only assert enter(m,s) for m,s such that depart(m,s) was already derived in another way and such that exit(m,s) was not derived. If we use the following algebraic constraint in the head

$$1 =_{\mathbb{N}} \text{enter}(M,S) * \neg\text{exit}(M,S)) * \text{depart}(M,S),$$

it can also derive depart(m,s) instead of using it as a precondition.

We can prevent this by making use of double negation and implication. Namely, we can instead use the following algebraic constraint

$$1 =_{\mathbb{N}} \neg\neg(\text{depart}(M,S) * \neg\text{exit}(M,S)) * ((\text{depart}(M,S) * \neg\text{exit}(M,S)) \rightarrow \text{enter}(M,S)).$$

In classical logic this constraint would be equivalent to the previous one: we could cancel the double negation and simplify the the implication as follows:

$$\begin{aligned}&\neg\neg(\text{depart}(M,S) \wedge \neg\text{exit}(M,S)) \wedge ((\text{depart}(M,S) \wedge \neg\text{exit}(M,S)) \rightarrow \text{enter}(M,S))\\ \equiv&\text{depart}(M,S) \wedge \neg\text{exit}(M,S) \wedge ((\text{depart}(M,S) \wedge \neg\text{exit}(M,S)) \rightarrow \text{enter}(M,S))\\ \equiv&\text{depart}(M,S) \wedge \neg\text{exit}(M,S) \wedge \text{enter}(M,S)\end{aligned}$$

In (weighted) HT Logic these simplifications are however not valid, since double negation does not cancel out. To see this, consider the formulas $\neg\neg\text{depart}(m,s)$ and $\text{depart}(m,s)$ with respect to the interpretation $(\emptyset, \{\text{depart}(m,s)\})$. While it is true that $(\emptyset, \{\text{depart}(m,s)\}, T) \models \neg\neg\text{depart}(m,s)$ and also $(\emptyset, \{\text{depart}(m,s)\}, T) \models \text{depart}(m,s)$ the equivalence does not hold for $(\emptyset, \{\text{depart}(m,s)\}, H)$. Namely, $(\emptyset, \{\text{depart}(m,s)\}, H) \not\models \text{depart}(m,s)$, since $\text{depart}(m,s)$ is not included in the interpretation Here, which is the empty set. On the other hand, $(\emptyset, \{\text{depart}(m,s)\}, H) \models \neg\neg\text{depart}(m,s)$, since by definition $\neg\neg\text{depart}(m,s) \equiv (\text{depart}(m,s) \rightarrow \bot) \rightarrow \bot$ and for the implications it holds that

$$\begin{aligned}&\mathcal{I}_H \models (\text{depart}(m,s) \rightarrow \bot) \rightarrow \bot\\ \Longleftrightarrow\ &\mathcal{I}_w \not\models \text{depart}(m,s) \rightarrow \bot \text{ or } \mathcal{I}_w \models \bot \text{ for all } w \in \{H,T\}\end{aligned}$$

Since $\mathcal{I}_w \not\models \bot$ this is equivalent to

$$\mathcal{I}_w \not\models \text{depart}(m,s) \rightarrow \bot \text{ for all } w \in \{H,T\},$$

which we can further expand as follows

$$\begin{aligned}&\mathcal{I}_w \not\models \text{depart}(m,s) \rightarrow \bot \text{ for all } w \in \{H,T\}\\ \Longleftrightarrow\ &\mathcal{I}_H \not\models \text{depart}(m,s) \rightarrow \bot \text{ and } \mathcal{I}_T \not\models \text{depart}(m,s) \rightarrow \bot\\ \Longleftrightarrow\ &\mathcal{I}_H \not\models \text{depart}(m,s) \rightarrow \bot \text{ and } \mathcal{I}_T \models \text{depart}(m,s) \text{ and } \mathcal{I}_T \not\models \bot\\ \Longleftrightarrow\ &\mathcal{I}_H \not\models \text{depart}(m,s) \rightarrow \bot\\ \Longleftrightarrow\ &\mathcal{I}_w \models \text{depart}(m,s) \text{ and } \mathcal{I}_w \not\models \bot \text{ for some } w \in \{H,T\}\\ \Longleftrightarrow\ &\mathcal{I}_w \models \text{depart}(m,s) \text{ for some } w \in \{H,T\}\end{aligned}$$

Since as we have seen before $\mathcal{I}_T \models \text{depart}(m,s)$ holds, the last expression evaluates to true, meaning that $\mathcal{I}_H \models (\text{depart}(m,s) \rightarrow \bot) \rightarrow \bot$ holds, even though $\text{depart}(m,s)$ is not included in the interpretation for Here.

This explains not only why double negation does not cancel out but also why the adapted algebraic constraint does not allow us to derive $\text{depart}(m,s)$ but uses it as a precondition: if $\text{depart}(m,s)$ does not hold in the There interpretation, then

$$[\![\neg\neg(\text{depart}(m,s) * \neg\text{exit}(m,s)) * ((\text{depart}(m,s) * \neg\text{exit}(m,s)) \rightarrow \text{enter}(m,s))]\!]_{\mathbb{N}}(\mathcal{I}_H)$$

evaluates to 0. However, if it does hold in the There interpretation, then it does not matter whether it is included in the Here interpretation. Since we are interested in

Equilibrium models, $\text{depart}(m, s)$ must therefore have been derived by another rule and only acts as a precondition in our constraint.

Note, however, that $\neg\neg\neg a$ is equivalent to $\neg a$ even in (weighted) HT Logic. Thus, we can optionally simplify our constraint to

$$1 =_{\mathbb{N}} \neg\neg\text{depart}(M, S) * \neg\text{exit}(M, S) * (\text{depart}(M, S) \to \text{enter}(M, S)).$$

More generally, we use the pattern

$$\alpha(\overline{X}, W) = \neg\neg\phi(\overline{X}, W) * (\phi(\overline{X}, W) \to p(\overline{X})) * W. \tag{2.33}$$

Abstractly, it ensures that $p(\overline{x})$ can only be asserted for $\overline{x}$ where we already know that $\phi(\overline{x}, w)$ holds, so we can not "invent" new constants or use the constraint to assert $\phi(\overline{x}, w)$.

The purpose of the variable W is to assign the addition of $p(\overline{x})$ a weight w.

This covers the case, where we want to derive a minimal or exact quantity of atoms. While such *minimized constraints* are useful, we also need to be able to specify *choice constraints* in our language. That is, we may want to specify that for any tuple $\overline{x}$ the atom $p(\overline{x})$ may hold, if $\phi(\overline{x})$ holds as a precondition.

This can be achieved naturally without extending the semantics of our language, by introducing a syntactic shorthand $k \sim^c_{\mathcal{R}} \alpha$ for *algebraic choice constraints* in rule-heads. We define that

$$r = k \sim^c_{\mathcal{R}} \alpha \leftarrow B(r) \quad \text{stands for} \quad k \sim_{\mathcal{R}} \alpha \leftarrow B(r) \;\text{ and }\; X =_{\mathcal{R}} \alpha \leftarrow X =_{\mathcal{R}} \alpha^{\neg\neg}, B(r), \tag{2.34}$$

where $\alpha^{\neg\neg}$ is obtained from α by adding $\neg\neg$ in front of each atom $p(\overline{x})$. These algebraic choice constraints behave as expected of choice constraints. (The next proposition considers only globally ground rules in order to decrease the amount of syntactic noise.)

Proposition 35 (Choice Semantics)**.** *For any σ-HT-interpretation $(\mathcal{I}_H, \mathcal{I}_T)$ and any globally ground rule $r = k \sim^c_{\mathcal{R}} \alpha \leftarrow B(r)$, it holds that $\mathcal{I}_H \models_\sigma r$ iff*

(i) *$\mathcal{I}_H \models_\sigma k \sim_{\mathcal{R}} \alpha \leftarrow B(r)$ and*

(ii) *if $\mathcal{I}_H \models_\sigma (B_\wedge(r))^\Sigma$, then $[\![(\alpha)^\Sigma]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_T) = [\![(\alpha)^\Sigma]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_H)$.*

Proof. According to our definition

$$\begin{aligned}
&\mathcal{I}_H \models_\sigma k \sim^c_{\mathcal{R}} \alpha \leftarrow B(r) \\
\iff &\mathcal{I}_H \models_\sigma k \sim_{\mathcal{R}} \alpha \leftarrow B(r) \text{ and } \mathcal{I}_H \models_\sigma X =_{\mathcal{R}} \alpha \leftarrow X =_{\mathcal{R}} \alpha^{\neg\neg}, B(r) \\
\iff &\mathcal{I}_H \models_\sigma k \sim_{\mathcal{R}} \alpha \leftarrow B(r) \text{ and } \mathcal{I}_H \models_\sigma \forall X\ X =_{\mathcal{R}} \alpha^{\neg\neg}, B(r) \to X =_{\mathcal{R}} \alpha
\end{aligned}$$

By the definition of $\alpha^{\neg\neg}$ we know that $[\![\alpha^{\neg\neg}]\!]_{\mathcal{R}}(\mathcal{I}_H)$ is equal to $[\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}_T)$. Therefore, we only need to consider the grounding of the rule where X is replaced by $[\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}_T)$. Then

$$\begin{aligned} & \mathcal{I}_H \models_\sigma r \\ \iff & \mathcal{I}_H \models_\sigma k \sim_{\mathcal{R}} \alpha \leftarrow B(r) \text{ and } \mathcal{I}_H \models_\sigma B(r) \to [\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}_T) =_{\mathcal{R}} \alpha \end{aligned}$$

□

Choice constraints are already well-known from previous ASP extensions, so we do not explain them in more detail. The usefulness of the novel minimized choice constraints is demonstrated in the following example.

Example 17 (Integer Subset Sum)**.** *Consider the following variation of the well-known Subset Sum Problem [Alf98]: Given a set $S \subseteq \mathbb{Z}$ and two bounds $l, u \in \mathbb{Z}$, determine a $\subseteq$-minimal solution $S' \subseteq S$ such that $l \leq \sum_{x \in S'} x \leq u$. This problem can be solved easily using ASP($\mathcal{AC}$) when $s(x)$ holds for $x \in S$:*

$$\begin{aligned} l \leq_{\mathbb{Z}} \neg\neg s(X) * (s(X) \to in(X)) * X \leftarrow \\ u \geq_{\mathbb{Z}} \neg\neg s(X) * (s(X) \to in(X)) * X \leftarrow \end{aligned}$$

For every equilibrium model $\mathcal{I}$ the set $S' = \{x \mid in(x) \in \mathcal{I}\}$ is a $\subseteq$-minimal solution and for every $\subseteq$-minimal solution S' there exists an equilibrium model $\mathcal{I}$ such that $S' = \{x \mid in(x) \in \mathcal{I}\}$.

Trying to use the choice constraint

$$k =^{c}_{\mathbb{Z}} \neg\neg s(X) \wedge (s(X) \to in(X)) \wedge X \leftarrow$$

instead, leads to solutions to the Subset Sum Problem but not necessarily to solutions of the Minimal Subset Sum Problem.

Aggregates As can be seen in Example 6, we can model aggregates whose aggregation function is the addition of some semiring. This restriction is mild in practice: The aggregates min, max, sum, count are expressible using a single algebraic constraint. times and avg are expressible using multiple algebraic constraints (e.g. avg is sum divided by count).

Value Guessing and Arithmetic Operators Value guessing and arithmetic operators are especially used in combinations of ASP and CP [Lie14]. We can guess a value from a semiring, perform arithmetic operations over semirings and evaluate (in)equations on the results. Again, we are mildly restricted as only semiring operations are available.

2.3.5 Complexity

We consider the computational complexity of the following problems:

- Model Checking (MC): Given a safe program Π and an interpretation $\mathcal{I}$ of Π, is $\mathcal{I}$ is an equilibrium model of Π?
- Satisfiability (SAT): Given a safe program Π, does Π have an equilibrium model?
- Strong Equivalence (SE): Given safe programs Π_1, Π_2, are Π_1 and Π_2 strongly equivalent?

Compared to ordinary ASP programs, the main aspect that complicates these problems is that we may have to evaluate weighted formulas over an arbitrary semiring. If we want to prevent an increase in complexity, then we need to encode the elements of the semiring in some way which allows for efficient calculations and comparison. To this end, we use efficient encodedness.

Definition 36 (Encoding Function, Efficiently Encoded Semiring)**.** *Let* $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ *be a semiring. Then an injective function* $e : R \to \{0,1\}^*$ *is an* encoding function.

Given an encoded value $e(r)$ *we define* $\|r\|_e$*, the* size of r w.r.t. e*, as the length of the bitstring* $e(r)$*, i.e.* $|e(r)|$.

Let $\mathcal{R}$ *be a semiring and* $e : R \to \{0,1\}^*$ *an encoding function. Then* $\mathcal{R}$ *is* efficiently encoded *by* e*, if there exists a polynomial* $p(x)$ *s.t. for all* $e(r_1), \ldots, e(r_n) \in e(R)$ *it holds that*

1. $\|\bigotimes_{i=1}^n r_i\|_e \leq p(n) \max_{i=1,\ldots,n} \|r_i\|_e$,
2. $\|\bigoplus_{i=1}^n r_i\|_e \leq p(\log_2(n)) \max_{i=1,\ldots,n} \|r_i\|_e$,
3. $\max(\|-r\|_e, \|r^{-1}\|_e) \leq \min(\|-r\|_e, \|r^{-1}\|_e) + p(0)$,
4. $e(r), e(r') \mapsto e(r \odot r')$ *is in FP for* $\odot = \oplus, \otimes$*, and*
5. $e(r) \mapsto e(f(r))$ *is in FP for* $f(.) = -(.), (.)^{-1}$.

This restriction is mild in practice; for example $\mathbb{B}$, $\mathbb{N}$, $\mathbb{Z}$, $\mathbb{Q}$, $\mathcal{R}_{\max,+}$, 2^A are efficiently encodable.

Theorem 37 (Ground Complexity)**.** *For variable-free programs over efficiently encoded semirings*

- *MC and (propositional) SE are co-NP-complete, and*
- *SAT is* Σ_2^p*-complete.*

Proof (Sketch, see Appendix A.4 for the full proof). The hardness parts are inherited from disjunctive logic programs [Dan+01], cf. Section 2.3.4, resp. HT-Logic [LPV01]. The membership parts result by guess and check algorithms: for similar bounds as in ordinary ASP, we just need that $\mathcal{I}_H \models k \sim_{\mathcal{R}} \alpha$ is polynomially decidable given $(\mathcal{I}^H, \mathcal{I}^T)$ and an algebraic constraint $k \sim_{\mathcal{R}} \alpha$; as $\mathcal{R}$ is efficiently encoded, this holds. □

The non-ground complexity is significantly higher.

Theorem 38 (Non-ground Complexity)**.** *Let* Π *be a safe program such that each semiring in* Π *is efficiently encoded. Then*

(i) *MC is in PSPACE and co-NP$^{NP^{PP}}$-hard, and*

(ii) *SAT and SE are undecidable.*

Proof. (i) Given the interpretation $\mathcal{I}$ (as set of ground atoms), we can iterate over all $\mathcal{I}' \subsetneq \mathcal{I}$ and check $(\mathcal{I}, \mathcal{I}, H) \models r'$, as well as $(\mathcal{I}', \mathcal{I}, H) \models r'$ for each ground instance r' of a rule $r \in \Pi$ in exponential time. The iteration and considering one ground instance r' at a time is feasible in polynomial space; the evaluation of algebraic constraints $k \sim_{\mathcal{R}} \alpha$ is feasible in polynomial space, since if α is of the form $\Sigma y_1, \ldots, y_n \alpha'(y_1, \ldots, y_n)$ where α' is quantifier-free, by safety of the program each y_i must occur in some atom $p(\overline{x})$. That is, to evaluate α, we only need to consider values $\xi(y_i)$ for y_i, $i = 1, \ldots, n$ that occur in the interpretation $\mathcal{I}$. There are exponentially many such ξ; for each of them, the value of $\alpha'(\xi(y_1), \ldots, \xi(y_n))$ can be computed in polynomial time given that $\mathcal{R}$ is efficiently encoded, yielding a value r_ξ such that $e(r_\xi)$ occupies polynomially many bits. The aggregation $\Sigma_\xi r_\xi$ over all ξ is then feasible in polynomial space by the assertion that $\|\bigoplus_{i=1}^n r_i\|_e \leq p(\log_2(n)) \max_{i=1,\ldots,n} \|r_i\|_e$ and that $e(r_1 \oplus r_2)$ is computable in polynomial time given $e(r_1), e(r_2)$.

The co-NP$^{\text{NP}^{\text{PP}}}$-hardness is due to a reduction from AE-MAJSAT [Wag86], which asks whether for a Boolean formula $\phi(x_1, \ldots, x_n)$ for all assignments to $x_1, \ldots, x_m$ a partial assignment to $x_{m+1}, \ldots, x_k$ exists s.t. more than 2^{n-k-1} of the assignments to $x_{k+1}, \ldots, x_n$ satisfy $\phi(\overline{x})$. Then the program

$$
\begin{aligned}
e(0) &\leftarrow \\
e(1) &\leftarrow \\
1 =_{\mathbb{B}} a_1(0) + a_1(1) &\leftarrow \\
&\ldots \\
1 =_{\mathbb{B}} a_m(0) + a_m(1) &\leftarrow \\
1 =_{\mathbb{B}} a_1(0) * a_1(1) &\leftarrow a_1(X_1), \ldots, a_m(X_m), e(X_{m+1}), \ldots, e(X_k), \\
&\qquad 2^{n-k-1} <_{\mathbb{N}} e(X_{k+1}) * \ldots * e(X_n) * \phi(\overline{X}) \\
&\ldots \\
1 =_{\mathbb{B}} a_m(0) * a_m(1) &\leftarrow a_1(X_1), \ldots, a_m(X_m), e(X_{m+1}), \ldots, e(X_k),
\end{aligned}
$$

$$2^{n-k-1} <_{\mathbb{N}} e(X_{k+1}) * \ldots * e(X_n) * \phi(\overline{X})$$

has an equilibrium model $\mathcal{I} = \{a_1(0), a_1(1), \ldots, a_m(0), a_m(1), e(0), e(1)\}$ iff the answer for AE-Majsat is yes.

Assume the answer for AE-Majsat is yes. Then for every subset $\mathcal{I}' \subseteq \mathcal{I}$ we have $(\mathcal{I}', \mathcal{I}, H) \not\models \Pi$. If we remove $e(i)$ or both $a_i(0)$ and $a_i(1)$ for some i, this is clear. Otherwise, we know that for each i some $a_i(j_i)$ holds. Then for these values there exist values $j_{m+1}, \ldots, j_k$ s.t. $\phi(j_1, \ldots, j_k, X_{k+1}, \ldots, X_n)$ is a yes instance for Majsat. Therefore the body

$$a_1(j_1), \ldots, a_m(j_m), e(X_{j+1}), \ldots, e(j_k), 2^{n-k-1} <_{\mathbb{N}} e(X_{k+1}) * \ldots * e(X_n) * \phi(\overline{X})$$

is satisfied and if $(\mathcal{I}', \mathcal{I}, H) \models \Pi$ we know that $\mathcal{I}' = \mathcal{I}$.

On the other hand if the answer for AE-Majsat is no, due to the partial assignment $j_1, \ldots, j_m$ to the variables $x_1, \ldots, x_m$, for $\mathcal{I}' = \{a_1(j_1), \ldots, a_m(j_m), e(0), e(1)\}$ we have $(\mathcal{I}', \mathcal{I}, H) \models \Pi$, and therefore $\mathcal{I}$ is not an equilibrium model.

(ii) The undecidable Mortal Matrix Problem (MMP) asks whether any product of matrices in $X = \{X_1, \ldots, X_n\} \subset \mathbb{Z}^{d \times d}$ evaluates to the zero matrix 0_d [Cas+14].

The semiring $(\mathbb{Z}^{d\times d}, +, \cdot, 0_d, 1_d)$ is efficiently encodable, and the program

$$\begin{aligned} p(X_i) &\leftarrow \qquad (i = 1, \ldots, n) \\ \bot &\leftarrow \neg p(0_d) \\ p(Y) &\leftarrow p(Z_1), p(Z_2), Y =_{\mathbb{Z}^{d\times d}} Z_1 \wedge Z_2 \end{aligned}$$

has an equilibrium model iff X is a yes-instance of MMP, as $p(0_d)$ needs to be supported. This shows undecidability of SAT.

For undecidability of SE, let Π be the program from above and $\Pi' = \Pi \setminus \{\bot \leftarrow \neg p(0)_d)\}$. As Π' has no negation, its HT-models are the interpretations $(\mathcal{I}', \mathcal{I})$ where both $\mathcal{I}'$ and $\mathcal{I}$ are closed under the rules of Π', sets S such that $p(X_1), \ldots, p(X_n) \in S$ and whenever $p(Y), p(Z) \in S$ then also $p(Y * Z) \in S$. Similarly, the HT-models of Π are the interpretations $(\mathcal{I}', \mathcal{I})$ where $\mathcal{I}'$ and $\mathcal{I}$ are closed under the rules of Π' and in addition $p(0_d) \in \mathcal{I}'$.

Therefore, $\Pi \equiv_s \Pi'$ iff $p(0_d) \in L$, where L is the least set closed under the rules of Π', which holds iff the answer for the mortal matrix problem on X is yes. □

As $\mathrm{NP}^{\mathrm{PP}}$ contains the polynomial hierarchy (PH), this places MC between PH and PSPACE.

For programs over the canonical semiring $\mathbb{N}$, MC is co-NP^C-complete for $C = \mathrm{NP}^{\mathrm{PP}}$ (while SAT and SE are undecidable).

Corollary 39. *Let Π be a safe program using only the semiring $\mathbb{N}$ of the natural numbers. Then we have co-$NP^{NP^{PP}}$-completeness for MC.*

Proof. The hardness is due to the proof of the previous theorem. The membership follows from the fact that we can check the satisfaction of constraints over $\mathbb{N}$ using a PP oracle.

This can be seen as follows. We can evaluate weighted formulas over $\mathbb{N}$ of the form $\Sigma y_1 \dots \Sigma y_n \alpha$ where α is quantifier-free using a #P oracle: we can non-deterministically choose an assignment ξ to $y_1, \dots, y_n$, calculate $r = [\![\alpha(\xi)]\!]_{\mathbb{N}}(\mathcal{I}_w)$ in polynomial time and generate r accepting branches.

Since P^{PP} is equal to $\mathrm{P}^{\#\mathrm{P}}$ also co-$\mathrm{NP}^{\mathrm{NP}^{\mathrm{PP}}}$ is equal to co-$\mathrm{NP}^{\mathrm{NP}^{\#\mathrm{P}}}$.

As for the co-$\mathrm{NP}^{\mathrm{NP}^{\#\mathrm{P}}}$ membership: Given a program Π and a potential equilibrium model $\mathcal{I}$ we can guess a subset $\mathcal{I}' \subsetneq \mathcal{I}$ and check whether $(\mathcal{I}', \mathcal{I}, H) \models \Pi$. The latter can be achieved in co-$\mathrm{NP}^{\#\mathrm{P}}$ by guessing a rule $r \in \Pi$ and an assignment ξ to its global variables. Then we can check whether $(\mathcal{I}', \mathcal{I}, H) \models r(\xi)$ in $\mathrm{P}^{\#\mathrm{P}}$ by checking satisfaction of each atom and constraint in $r(\xi)$. □

We note that while MC is decidable for $\mathcal{AC}$-programs over the natural numbers, SAT and SE are undecidable. This may not be very surprising given Theorem 38 and the fact that the semiring $\mathbb{Z}^{d\times d}$ is similar to $\mathbb{N}$. The undecidability can directly be shown by a reduction from solving Diophantine equations, i.e., polynomial equations $P(x_1, ..., x_n) = 0$ in variables $x_1, \dots, x_n$ over the integers, which by Matiyasevich's celebrated result is undecidable; this holds if the solutions are restricted to the natural numbers [Mat96]. We can equivalently consider polynomial equations $P(x_1, ..., x_n) = Q(x_1, ..., x_n)$ where all coefficients in the polynomial expressions $P(x_1, \dots, x_n)$ and $Q(x_1, \dots, x_n)$ are non-negative. We then can write a program Π consisting of the rules

$$\begin{aligned} \mathrm{n}(0) &\leftarrow . \\ \mathrm{n}(X) &\leftarrow \mathrm{n}(Y), X =_{\mathbb{N}} 1 + Y. \\ \mathrm{sol} &\leftarrow \mathrm{n}(X_1), ..., \mathrm{n}(X_n), Y =_{\mathbb{N}} P(X_1, ..., X_n), Y =_{\mathbb{N}} Q(X_1, ..., X_n). \\ \bot &\leftarrow \neg\mathrm{sol}. \end{aligned}$$

The program Π is safe and it has a (unique) equilibrium model (in which sol is true) iff a solution to $P(x_1, ..., x_n) = Q(x_1, ..., x_n)$ exists. Furthermore, the existence of an equilibrium model is equivalent to $\Pi \equiv_s \Pi \setminus \{\bot \leftarrow \neg\mathrm{sol}\}$.

We can retain decidability for SAT and SE when the programs considered are finitely groundable.

Definition 40 (Finite Groundability)**.** *Let σ be some semiring signature and let Π be an $\mathcal{AC}$-program over σ. Then Π is* finitely groundable *if there is a signature $\sigma' = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$, where $\mathcal{D}$ is finite, such that the equilibrium models of Π over σ are the same as the equilibrium models over σ'.*

Let us call a semiring *computable* if all operators $\oplus, \otimes, -, ^{-1}, >$ are computable.

Theorem 41. *For finitely groundable programs over $\sigma' = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ (where $\mathcal{D}$ is finite) that only use computable semirings, SAT and SE are decidable.*

Proof. In the general case, we can eliminate quantifiers by replacing universally quantified formulas with finite conjunctions over all the substitutions and existentially quantifies formulas with finite disjunctions over all the substitutions.

For ground programs we have decidability when all the semirings are computable. □

We can ensure finite groundability by limiting value invention, i.e. constraints of the form $X =_{\mathcal{R}} \alpha(\overline{Y})$, and value guessing. For the latter, we adapt domain restrictedness from [NSS99].

Definition 42 (Domain Restrictedness)**.** *An algebraic constraint is* domain restricted *in variables $\overline{X}$, if it is of the form*

$$k \sim_{\mathcal{R}} \neg\neg\alpha(\overline{X}) * (\alpha(\overline{X}) \to \beta(\overline{X})) * \gamma(\overline{X}),$$

where $\alpha(\overline{X}), \beta(\overline{X})$ are syntactically domain independent and all atoms in $\gamma(\overline{X})$ are locally ground.

Intuitively, only constants "known" by predicates in α can be "transferred" to predicates in β, and γ assigns a weight to each substitution. The pattern is explained less generally in Section 2.3.4.

Theorem 43. *Let Π be a safe program over σ without value invention, where all algebraic constraints in heads are domain restricted. Then Π is finitely groundable over $\sigma' = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$, where $\mathcal{D}$ is the subset of domain values that occur in Π.*

Proof. Let $\mathcal{I}$ be a σ-interpretation s.t. $(\mathcal{I}, \mathcal{I}, T) \models \Pi$. Then for $\mathcal{I}'$ obtained from $\mathcal{I}$ by removing all atoms that contain constants not from $\mathcal{D}$, we have $(\mathcal{I}', \mathcal{I}, H) \models \Pi$. This can be seen as follows: Assume $r \in \Pi$ with global variables $x_1, \ldots, x_n$. Since r is safe and does not contain value invention, the body of r can only be satisfied for substitutions of the variables with elements from $\mathcal{D}$. Therefore, if the head of r is an atom $p(\overline{x})$, we can only derive $p(\overline{\xi})$ for substitutions from $\mathcal{D}$ and therefore, if the rule was satisfied previously, it is still satisfied.

Otherwise, if the head of r is a constraint, we know that it is domain restricted, i.e. of the form

$$k \sim_{\mathcal{R}} \neg\neg\alpha(\overline{X}) * (\alpha(\overline{X}) \to \beta(\overline{X})) * \gamma(\overline{X}),$$

where $\alpha(\overline{X}), \beta(\overline{X})$ are syntactically domain independent and all atoms in $\gamma(\overline{X})$ are locally ground.

Let $\overline{\xi}$ be some assignments to $\overline{X}$ over the original domain.

We consider first $\neg\neg\alpha(\overline{\xi})$. It holds that

$$[\![\neg\neg\alpha(\overline{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_H) = e_{\oplus} \iff [\![\neg\alpha(\overline{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_H) = e_{\otimes} \vee [\![\neg\alpha(\overline{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_T) = e_{\otimes}$$

$$\iff [\![\alpha(\bar{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_H) = e_\oplus \wedge [\![\alpha(\bar{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_T) = e_\oplus \vee [\![\alpha(\bar{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_T) = e_\oplus$$
$$\iff [\![\alpha(\bar{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_T) = e_\oplus$$

Therefore this part of the formula is not influenced by $\mathcal{I}'$.

Secondly we consider $\alpha(\bar{\xi}) \to \beta(\bar{\xi})$. We only need to consider this value if $[\![\neg\neg\alpha(\bar{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_H)$ is unequal to zero, i.e. if $[\![\alpha(\bar{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_T)$ is unequal to zero. Now if $[\![\alpha(\bar{\xi}) \to \beta(\bar{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_T)$ is unequal to zero this implies that $[\![\beta(\bar{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_T)$ is unequal to zero. If all the values in $\bar{\xi}$ are from $\mathcal{D}$, there is no change. Otherwise we know that $[\![\alpha(\bar{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_H) = [\![\beta(\bar{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_H) = e_\oplus$ since $\alpha(\overline{X})$ and $\beta(\overline{X})$ are syntactically domain independent and therefore have value $e_\oplus$ for values that are not mentioned in the interpretation $\mathcal{I}'$ (see proof of the invariance of the support for syntactically domain independent weighted formulas). It follows that $[\![\alpha(\bar{\xi}) \to \beta(\bar{\xi})]\!]_{\mathcal{R}}(\mathcal{I}_T)$ is also unequal to zero.

Since $\gamma(\overline{X})$ contains only locally ground atoms, the restriction of the interpretation to $\mathcal{I}'$ does not change the value of $\gamma(\bar{\xi})$.

Therefore

$$[\![\neg\neg\alpha(\overline{X}) * (\alpha(\overline{X}) \to \beta(\overline{X})) * \gamma(\overline{X})]\!]_{\mathcal{R}}(\mathcal{I}_H) = [\![\neg\neg\alpha(\overline{X}) * (\alpha(\overline{X}) \to \beta(\overline{X})) * \gamma(\overline{X})]\!]_{\mathcal{R}}(\mathcal{I}_T)$$

and

$$\mathcal{I}_H \models k \sim_{\mathcal{R}} \neg\neg\alpha(\overline{X}) * (\alpha(\overline{X}) \to \beta(\overline{X})) * \gamma(\overline{X})$$
$$\iff \mathcal{I}_T \models k \sim_{\mathcal{R}} \neg\neg\alpha(\overline{X}) * (\alpha(\overline{X}) \to \beta(\overline{X})) * \gamma(\overline{X}).$$

We see that since $(\mathcal{I}, \mathcal{I}, T) \models \Pi$ also $(\mathcal{I}', \mathcal{I}, T) \models \Pi$. Therefore $\mathcal{I}$ can only be an equilibrium model if it contains only constants from $\mathcal{D}$, which implies that Π is finitely groundable over σ' □

It follows that we can use domain restrictedness to retain decidability.

Theorem 44. *Let Π be a safe program without value invention, where all algebraic constraints in rule heads are domain restricted. If all semirings in Π are computable, then both SAT and SE are decidable.*

Proof. This result follows easily from Theorems 41 and 43. □

However, prohibiting value invention entirely is unnecessarily strong. Weaker restrictions like aggregate stratification [FPL11] or argument restrictedness [LL09] can be adapted to ASP($\mathcal{AC}$). The resulting programs are also finitely groundable [Cal+08a] and therefore decidable.

2.3.6 Summary & Open Issues

We have seen that algebraic constraints unify many previously proposed constructs for more succinct answer set programs, with low practical restrictions and no increase in the ground complexity. Thus, algebraic constraints satisfy our requirements for being a general uniform extension that allows for succinct specifications of quantitative constraints. Even further than that, we can specify whether constraints in rule-heads are minimized or guessed, can explicitly represent values from different sets and give an interesting alternative semantics for conditionals.

We currently consider only a fragment of the weighted formulas. It would be interesting to see in the future, if other new and useful constructs can be expressed with a different fragment. Furthermore, an in-depth study of suitable conditions for finite groundability and the non-ground complexity in this context are indispensable for possible future work on an implementation. For this purpose an interesting problem needs to be solved: On the one hand, we do not want to restrict the expressivity of our framework but, on the other hand, we need to find conditions to ensure that a priori that we only need to consider a finite, preferably small domain. Work from the field of finite groundability [LL09; BL10] of programs with function symbols might be of use.

In terms of our main goal of finding a general framework for quantitative and temporal reasoning, algebraic constraints are not only sufficiently general. They are also promising, when it comes to an integration with algebraic measures and a temporal domain. We can use the general applicability of HT and Weighted Logic and combine ASP($\mathcal{AC}$) with temporal reasoning and algebraic measures, which should be much simpler for ASP($\mathcal{AC}$) than for others since both Weighted and HT logic facilitate generalization. Furthermore, since algebraic constraints and algebraic measures both make use of weighted logic, their integration should go rather smoothly without much additional effort.

2.4 Combining Stream Reasoning and Quantitative Reasoning

In the previous two sections, we introduced two general and uniform extensions of ASP that allow us to express quantitative reasoning tasks over the set of answer sets and quantitative constraints that enable the succinct specification of answer sets. These extensions are algebraic measures and algebraic constraints, respectively. Recalling our initial goal, we see that we need a general framework that extends ASP to the temporal domain and subsequently its integration with algebraic measures and algebraic constraints.

There already exist general frameworks for temporal reasoning with answer set semantics, among them LARS [BDE18], a Logic-based Framework for Analytic Reasoning over Streams, Temporal Equilibrium Logic (TEL) [CV07; Cab+18], and Metric Temporal Equilibrium Logic (MEL) [Cab+20a]. All of these logics provide a suitable basis for an general extension combining temporal and quantitative reasoning.

TEL and MEL have the benefit of being defined as Equilibrium Logic on the basis of Here-and-There semantics, which we also used in the definition of algebraic constraints, whereas the answer set semantics of LARS is defined using the so called FLP-reduct [FLP04]. While these semantics align on normal answer set programs, this is not the case in general. On the other hand, LARS has access to general window functions, which neither TEL and MEL have. Window functions are intuitively interpretation modifiers that filter out some of the true atoms in an interpretation. They allow LARS to capture not only many of the temporal operators from MEL and TEL but also more advanced features that cannot be expressed in MEL or TEL as succinctly. We, therefore, use LARS as the basis for our extension and address the issue of the difference in non-monotonicity by using a previously identified fragment of LARS [BDE16], where HT-semantics is defined and aligns with the standard semantics.

In Section 2.4.1 we first introduce the standard semantics for LARS, recall the fragment LARS_{HT}, where it aligns with HT-semantics by restating the HT-semantics in terms of sorted first-order HT Logic augmented with window functions. This enables the integration of LARS_{HT} with algebraic constraints as well as algebraic measures by deriving the corresponding weighted version over semirings of sorted first-order HT Logic augmented with window functions in Section 2.4.2. We follow up the definitions by an extensive example, before we analyze the ground expressivity and complexity of LARS measures without algebraic constraints in Section 2.4.3 and Section 2.4.4, respectively. Finally, we conclude with a discussion and future perspectives in Section 2.4.5.

2.4.1 LARS

LARS [BDE18] is a stream reasoning framework, in which the observed data may vary between the different time points in the considered, discrete interval. As previously, we consider a first-order context, meaning we assume a set $\mathcal{P}$ of predicates, domain $\mathcal{D}$ and set $\mathcal{X}$ of variables to be given. Then, an atom is of the form $p(t_1, \ldots, t_n)$ for $p \in \mathcal{P}$, $t_i \in \mathcal{D} \cup \mathcal{X}$, and $n = \text{arity}(p)$, the arity of the predicate p. It is variable free if $t_i \in \mathcal{D}$ for all $i = 1, \ldots, n$. As before, we denote by $\mathbf{HB}(\mathcal{P}, \mathcal{D})$ the set of all atoms over predicates in $\mathcal{P}$ and domain $\mathcal{D}$. Furthermore, we use $\mathbf{HB}$ as a shorthand for a Herbrand base $\mathbf{HB}(\mathcal{P}, \mathcal{D})$, where $\mathcal{P}$ and $\mathcal{D}$ are left implicit.

Interpretations of LARS formulas are formalized by *streams* S, which are pairs (T, v), where $T = \{t, t+1, \ldots, t+n\} \subseteq \mathbb{N}$ is the finite interval of discrete timepoints from t to $t+n$, for some $n \in \mathbb{N}$, and $v : T \to 2^{\mathbf{HB}}$ expresses that at time $t' \in T$ the atoms $v(t')$ appear in the stream. We say a stream $S' = (T', v')$ is a substream of S, if $T' \subseteq T$ and for all $t' \in T' : v'(t') \subseteq v(t')$, denoted $S' \subseteq S$.

Definition 45 (Window Function)**.** *A* window function ϖ *given a stream* S *and time point* t *restricts* S *to a substream* $\varpi(S, t) \subseteq S$*.*

Intuitively such window functions filter the data in a stream and give a snapshot of the data.

Example 18 (Window Function)**.** *An example of a window function, is given by* $around_n$*, where for a stream* $S = (T, v)$ *and* $t \in T$

$$\begin{aligned} around_n(S,t) &= (T', v'), \\ T' &= T \cap \{t-n, t-n+1, \ldots, t+n\}, \\ v'(t') &= v(t') \qquad\qquad \text{for } t' \in T'. \end{aligned}$$

I.e., $around_n$ *restricts a stream to only* n *time points around* t *on both sides. Similarly, the window functions future and past restrict a stream to the future and past time points, respectively.*

The above example is very specific, whereas the definition of window functions is very general: it allows literally any function, as long as it always returns a substream of the stream given as input. Other examples include restrictions to the data occurring in the last three time points or the latest 20 atoms.

LARS formulas are defined by the grammar

$$\alpha ::= p \mid \neg\alpha \mid \alpha \wedge \alpha \mid \alpha \vee \alpha \mid \Diamond\alpha \mid \Box\alpha \mid @_t\alpha \mid \boxplus^{\varpi}\alpha \mid \triangleright\alpha,$$

where $t \in \mathbb{N} \cup \mathcal{X}$, p is an atom and ϖ is a window function. A LARS formula is *variable free* if all of its atoms are variable free.

Here, $\triangleright$ is the reset operator, which resets a stream $S \subseteq S^\star$ obtained by applying window functions(s) to the original stream $S^\star$. It allows for more succinct program specifications. We omit non-temporal quantifiers and use grounding semantics to introduce the standard LARS semantics.

In the standard semantics, we use the shorthands $\bot := p \wedge \neg p$, $\top := \neg\bot$ and $\alpha \to \beta := \neg\alpha \vee \beta$. A *(pointed) LARS interpretation* is a tuple $\mathcal{I} = (S^\star, S, t)$ of streams $S^\star = (T^\star, v^\star), S = (T, v)$ and a time point $t \in T^\star$, where $S \subseteq S^\star$. *Satisfaction* of a variable free LARS formula α by $\mathcal{I}$, in symbols $\mathcal{I} \models \alpha$, is inductively defined by

$$\begin{array}{lcl}
\mathcal{I} \models p & \iff & p \in v(t), \text{ for } p \in \mathbf{HB} \\
\mathcal{I} \models \neg\alpha & \iff & \mathcal{I} \not\models \alpha \\
\mathcal{I} \models \alpha \wedge \beta & \iff & \mathcal{I} \models \alpha \text{ and } \mathcal{I} \models \beta \\
\mathcal{I} \models \alpha \vee \beta & \iff & \mathcal{I} \models \alpha \text{ or } \mathcal{I} \models \beta \\
\mathcal{I} \models \Diamond\alpha & \iff & \exists t' \in T : (S^\star, S, t') \models \alpha \\
\mathcal{I} \models \Box\alpha & \iff & \forall t' \in T : (S^\star, S, t') \models \alpha \\
\mathcal{I} \models @_{t'}\alpha & \iff & (S^\star, S, t') \models \alpha \text{ and } t' \in T \\
\mathcal{I} \models \boxplus^{\varpi}\alpha & \iff & (S^\star, \varpi(S,t), t) \models \alpha \\
\mathcal{I} \models \triangleright\alpha & \iff & (S^\star, S^\star, t) \models \alpha
\end{array}$$

We identify the pair (S, t) with the LARS interpretation (S, S, t).

Definition 46 (LARS Program)**.** *A* LARS program *is a finite set* Π *of rules* r *of the form* $r = \alpha \leftarrow \beta$*, where* α, β *are LARS formulas. We may write* α *and* $\neg\beta$ *as a shorthand for* $\alpha \leftarrow \top$ *and* $\bot \leftarrow \beta$*, respectively.*

The grounding *ground*(Π) *of a program* Π *over* $\mathcal{P}, \mathcal{V}$ *and* $\mathcal{D}$ *is given by the set*

$$ground(\Pi) = \{r\sigma \mid r \in \Pi, \sigma : \mathcal{V} \to \mathcal{D}\},$$

where σ *is called a substitution and* $r\sigma$ *denotes the result of applying* σ *to* r*, which means that every variable* V *that occurs in* r *is replaced by* $\sigma(V)$*.*

A rule $r = \alpha \leftarrow \beta$ for variable free α and β, is satisfied at a time point t by a stream S (written $(S,t) \models r$), if $(S,t) \models \beta \rightarrow \alpha$. Programs are seen as the conjunction of the rules in their grounding; thus, a stream S satisfies a program Π at time t (written $(S,t) \models \Pi$), if for all $r \in$ ground(Π), it holds that $(S,t) \models r$.

We distinguish between *extensional* and *intensional* atoms contained in $\mathbf{HB}^\epsilon$ and $\mathbf{HB}^I$ respectively. Extensional atoms represent the input data (and therefore do not occur in rule heads).

Definition 47 (Data and Interpretation Stream)**.** *A* data stream *is a stream* $D = (T, v)$ *where* v *asserts only extensional atoms, i.e.* $\forall t \in T : v(t) \subseteq \mathbf{HB}^\epsilon$*. A stream* $S = (T, v')$ *is an* interpretation stream *of* D*, if* $D \subseteq S$ *and* v, v' *agree on* $\mathbf{HB}^\epsilon$*.*

Finally, the standard answer sets semantics is defined using the FLP-*reduct* [FLP04], which for a program Π with respect to a pair (S,t), is given by

$$\Pi^{S,t} = \{\alpha \leftarrow \beta \in \Pi \mid (S,t) \models \beta\}.$$

Definition 48 (Answer Stream)**.** *An interpretation stream* S *of* D *is an* answer stream *for* Π *at* t*, if* (S,t) *satisfies* Π *and no interpretation stream* $S' \subsetneq S$ *of* D *exists s.t.* (S',t) *satisfies* $\Pi^{S,t}$*. We denote the set of such streams by* $\mathcal{AS}(\Pi, D, t)$*.*

Example 19 (Metro Connections)**.** *In the original metro connections example, we assumed the predicate depart to be given in the form of input facts. It is, however, not completely realistic in most cases that we know exactly when a given metro leaves. Instead, we usually know a scheduled departure time and it is likely that the metro leaves* around *that time.*

Assuming we are given the metro schedule as input, we can model that there are different possibilities for the actual departure time around the scheduled time by using the following rule:

$$@_T \boxplus^{around_3} \Diamond depart(M,S) \leftarrow @_T scheduled(M,S) \tag{2.35}$$

Here, the at operators $@_T$ with variable time T are intuitively the same as universally quantifying the rule

$$\boxplus^{around_3}\Diamond depart(M,S) \leftarrow scheduled(M,S) \tag{2.36}$$

over the temporal domain.

Note that in LARS the input predicates are explicitly separated from the rest: they are the extensional atoms in $\mathbf{HB}^{\epsilon}$.

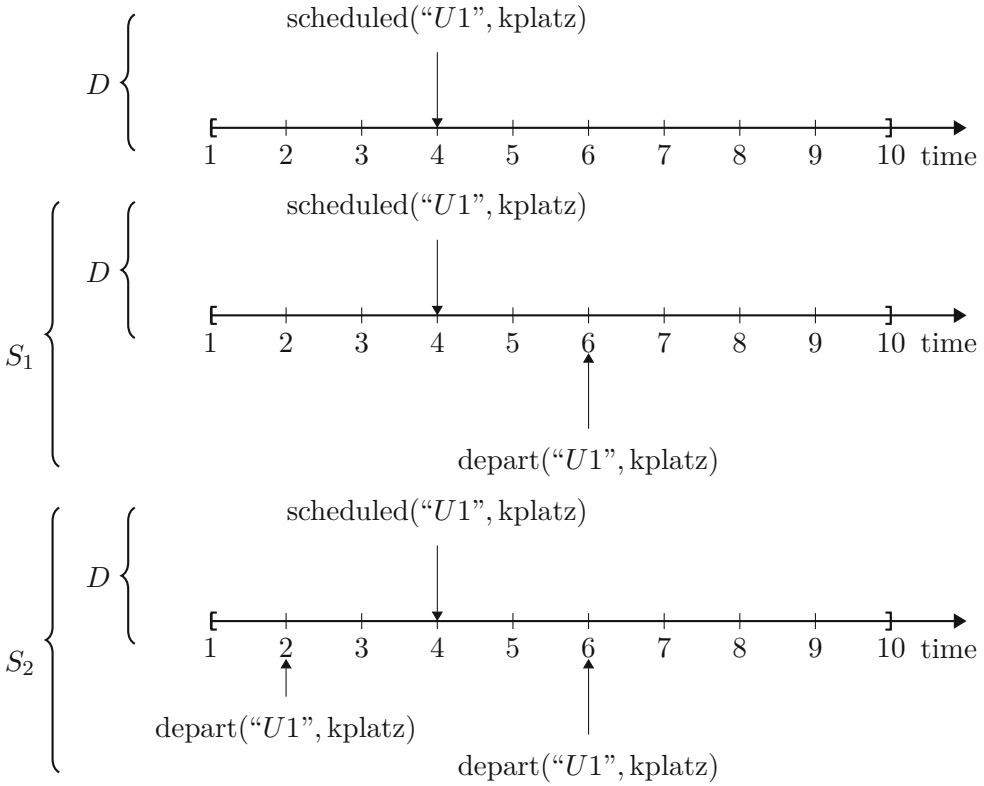


Figure 2.1: Different streams S_1, S_2 that extend a data stream D in the metro connections example.

For illustration purposes, we consider the answer streams of the above rule given the data stream $D = (T, v)$, where $T = \{1, 2, \ldots, 10\}$ and $v(t) = \{scheduled(\text{"}U1\text{"}, kplatz)\}$, if $t = 4$ and $v(t) = \emptyset$, otherwise. It is graphically illustrated as the top-most stream in Figure 2.1. Since $scheduled(\text{"}U1\text{"}, kplatz)$ holds at time $t = 4$, we need to satisfy also $\boxplus^{around_3}\Diamond depart(\text{"}U1\text{"}, kplatz)$ at time $t = 4$. This implies that $depart(\text{"}U1\text{"}, kplatz)$ must hold during at least one time point in $\{1, 2, \ldots, 7\}$. Therefore, D is not an answer stream. However, S_1 and S_2 in Figure 2.1 satisfy this requirement.

Furthermore, since we are not interested in streams that only satisfy the rule but are answer streams of it, there cannot be more than one time point, where $depart(\text{"}U1\text{"}, kplatz)$

holds. Accordingly, out of the streams S_1 and S_2 in Figure 2.1, only S_1 is an answer stream, since S_2 is not minimal.

For more background on LARS and a study of properties see [BDE18].

LARS$_{HT}$

As mentioned before, the standard semantics for LARS is unsuitable for an integration with algebraic constraints, which are defined using Here-and-There semantics. We therefore consider LARS$_{HT}$, which is LARS under Here-and-There semantics as in [BDE16] and show equivalence to the standard semantics on a fragment of LARS.

Algebraic constraints are defined for sorted first-order Here-and-There Logic. Since we want to integrate LARS$_{HT}$ and algebraic constraints, we do not introduce Here-and-There semantics directly for LARS formulas but extend the semantics of sorted first-order Here-and-There Logic to include window functions and assign LARS formulas a semantics via translation this logic.

This way we do not have to handle first-order and temporal quantifiers explicitly. Furthermore, it allows us to give a simpler extension to semiring weighted logics in the next section.

We start by adding window functions to our logic.

Definition 49 (Timed Signature, Syntax)**.** *A signature $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ is a* timed signature*, if*

1. *a distinguished time sort $\mathcal{T}$ is in $\mathcal{S}$,*
2. *$\mathcal{D}$ contains $\mathbb{N}$,*
3. *$\mathfrak{r} : \mathcal{S} \to 2^{\mathcal{D}}$ maps $\mathcal{T}$ to $\mathbb{N}$, and*
4. *a distinguished time interval predicate T is in $\mathcal{P}$.*

timed *σ-formulas are of the form*

$$\phi ::= \bot \mid p(\overline{x}, t) \mid \phi \to \phi \mid \phi \vee \phi \mid \phi \wedge \phi \mid \exists y \phi \mid \forall y \phi \mid \boxplus^{\varpi, t} \phi \mid \triangleright \phi, \tag{2.37}$$

where $p \in \mathcal{P}$, $\overline{x} = x_1, \ldots, x_n$, $x_i \in \mathcal{D} \cup \mathcal{X}$, $t \in \mathbb{N}$ or $s(t) = \mathcal{T}$, $y \in \mathcal{X}$, and ϖ is a window function; $p(\overline{x}, t)$ is called a σ-atom*. We define $\neg\phi$ as $\phi \to \bot$. A (timed)* σ-sentence *is a (timed) σ-formula without free variables.*

To apply window functions on σ-interpretations, we need to ensure that they correspond to streams. We thus introduce timed σ-interpretations and formalize the application of window functions on them.

Definition 50 (Timed σ-interpretation, Associated Stream/Interpretation)**.** *Let $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ be a timed signature. A* timed *σ-interpretation is a σ-interpretation $\mathcal{I}$ s.t.*

$$T(\mathcal{I}) = \{t \in \mathcal{D} \mid T(t) \in \mathcal{I}\},$$

the time interval *of $\mathcal{I}$, is of the form $T(\mathcal{I}) = \{t, t+1, \ldots, t+n\}$ for some $t, n \in \mathbb{N}$.*

Given a timed σ-interpretation $\mathcal{I}$, the associated stream $S(\mathcal{I}) = (T(\mathcal{I}), v(\mathcal{I}))$, where

$$v(\mathcal{I})(t) = \{p(\overline{x}) \mid p(\overline{x}, t) \in \mathcal{I}\}$$

is the valuation function *of $\mathcal{I}$.*

Similarly, given a stream $S = (T, v)$, the associated timed σ-interpretation $\mathcal{I}(S)$ is given by

$$\mathcal{I}(S) = \{p(\overline{x}, t) \mid p(\overline{x}) \in v(t), t \in T\} \cup \{T(t) \mid t \in T\}.$$

Naturally, when we want to apply a window function ϖ at time t to a timed σ-interpretation $\mathcal{I}$, we instead apply it the associated stream $S(\mathcal{I})$, resulting in $\varpi(S(\mathcal{I}), t)$, and take the associated timed σ-interpretation, resulting in

$$\varpi(\mathcal{I}) = \mathcal{I}(\varpi(S(\mathcal{I}), t)).$$

Using this correspondence it becomes clear how we can extend the HT Semantics to the extended logic.

Definition 51 (Timed HT-σ-interpretation, HT Semantics)**.** *Let $\mathcal{I}^{H*}, \mathcal{I}^{H}, \mathcal{I}^{T*}, \mathcal{I}^{T}$ be timed σ-interpretations, s.t. $\mathcal{I}^{H} \subseteq \mathcal{I}^{T}$ and $\mathcal{I}^{H*} \subseteq \mathcal{I}^{T*}$. Then $(\mathcal{I}^{*}, \mathcal{I}) = (\mathcal{I}^{H*}, \mathcal{I}^{T*}, \mathcal{I}^{H}, \mathcal{I}^{T})$ is a* timed *σ-HT-interpretation and $\mathcal{I}_w = (\mathcal{I}^{*}, \mathcal{I}, w)$, for $w \in \{H, T\}$, is a* timed *pointed σ-HT-interpretation.*

Satisfaction of a timed σ-sentence ϕ with respect to a timed pointed σ-HT-interpretation $\mathcal{I}_w = (\mathcal{I}^{H*}, \mathcal{I}^{T*}, \mathcal{I}^{H}, \mathcal{I}^{T}, w)$ is defined as follows, where we have the reflexive order $\geq$ on $\{H, T\}$, with $T \geq H$:

$$\begin{aligned}
\mathcal{I}_w &\not\models_\sigma \bot \\
\mathcal{I}_w &\models_\sigma p(\overline{x}) && \iff && p(\overline{x}) \in \mathcal{I}^{w*} \\
\mathcal{I}_w &\models_\sigma \phi \to \psi && \iff && \mathcal{I}_{w'} \not\models_\sigma \phi \text{ or } \mathcal{I}_{w'} \models_\sigma \psi \text{ for all } w' \geq w \\
\mathcal{I}_w &\models_\sigma \phi \vee \psi && \iff && \mathcal{I}_w \models_\sigma \phi \text{ or } \mathcal{I}_w \models_\sigma \psi \\
\mathcal{I}_w &\models_\sigma \phi \wedge \psi && \iff && \mathcal{I}_w \models_\sigma \phi \text{ and } \mathcal{I}_w \models_\sigma \psi \\
\mathcal{I}_w &\models_\sigma \exists x \phi(x) && \iff && \mathcal{I}_w \models_\sigma \phi(\xi), \text{ for some } \xi \in \mathfrak{r}(s(x)) \\
\mathcal{I}_w &\models_\sigma \forall x \phi(x) && \iff && \mathcal{I}_w \models_\sigma \phi(\xi), \text{ for all } \xi \in \mathfrak{r}(s(x)) \\
(\mathcal{I}^{H*}, \mathcal{I}^{T*}, \mathcal{I}^{H}, \mathcal{I}^{T}, w) &\models_\sigma \boxplus^{\varpi,t} \phi && \iff && (\mathcal{I}^{H*}, \mathcal{I}^{T*}, \varpi(\mathcal{I}^{H}, t), \varpi(\mathcal{I}^{T}, t), w) \models_\sigma \phi \\
(\mathcal{I}^{*}, \mathcal{I}, w) &\models_\sigma \triangleright \phi && \iff && (\mathcal{I}^{*}, \mathcal{I}^{*}, w) \models_\sigma \phi
\end{aligned}$$

When T is a set of σ-sentences, then $\mathcal{I}_w \models_\sigma T$ if $\forall \phi \in T : \mathcal{I}_w \models_\sigma \phi$.

Clearly, this does not define a semantics for LARS formulas. However, we can obtain an HT semantics for LARS formulas, by using the following embedding of LARS formulas into σ-formulas.

Definition 52 (LARS_{HT}-Embedding)**.** *We define the* LARS_{HT}*-embedding* $\mathcal{E}$ *for a LARS formula* α *and time point* $t \in \mathbb{N}$ *or time variable* $t \in \mathcal{X}$*, s.t.* $s(t) = \mathcal{T}$*, via structural induction on the formula.*

$$\begin{aligned}
\mathcal{E}(p(\overline{x}), t) &= p(\overline{x}, t) \\
\mathcal{E}(\neg\alpha, t) &= \neg\mathcal{E}(\alpha, t) \\
\mathcal{E}(\alpha \wedge \beta, t) &= \mathcal{E}(\alpha, t) \wedge \mathcal{E}(\beta, t) \\
\mathcal{E}(\alpha \vee \beta, t) &= \mathcal{E}(\alpha, t) \vee \mathcal{E}(\beta, t) \\
\mathcal{E}(\Diamond\alpha, t) &= \exists t' T(t') \wedge \mathcal{E}(\alpha, t') \\
\mathcal{E}(\Box\alpha, t) &= \forall t' \neg T(t') \vee \mathcal{E}(\alpha, t') \\
\mathcal{E}(@_{t'}\alpha, t) &= T(t') \wedge \mathcal{E}(\alpha, t') \\
\mathcal{E}(\boxplus^{\varpi}\alpha, t) &= \boxplus^{\varpi, t}\mathcal{E}(\alpha, t) \\
\mathcal{E}(\triangleright\alpha, t) &= \triangleright\mathcal{E}(\alpha, t)
\end{aligned}$$

Here, t' *is a new variable with* $s(t') = \mathcal{T}$*.*

The LARS_{HT} semantics is then simply given by the semantics of the embedded formula over a suitable signature.

Definition 53 (LARS_{HT} Semantics)**.** *Let* Π *be a LARS program over* $\mathcal{P}, \mathcal{V}, \mathcal{D}$ *and let* $var(r)$ *denote the variables of a LARS rule* r*. Furthermore, let* σ *be the timed signature* $\langle \mathcal{D}, \mathcal{P}, \mathcal{V}, \mathcal{S}, \mathfrak{r} \rangle$*. Then, given pointed LARS interpretations* (S^{H*}, S^H, t) *and* (S^{T*}, S^T, t)*, where* $S^{w*} = (T^{w*}, v^{w*})$ *and* $S^w = (T^w, v^w)$ *for* $w \in \{H, T\}$*, the satisfaction of* Π *with respect to the pointed* $LARS_{HT}$*-interpretation* $(S^{H*}, S^H, S^{T*}, S^T, w, t)$ *is defined by*

$$(S^{H*}, S^H, S^{T*}, S^T, w, t) \models \Pi$$
$$\iff$$
$$(\mathcal{I}(\mathcal{S}^{H*}), \mathcal{I}(\mathcal{S}^{T*}), \mathcal{I}(\mathcal{S}^H), \mathcal{I}(\mathcal{S}^T), w) \models_\sigma \bigwedge_{\alpha \leftarrow \beta \in \Pi} \forall_{x \in var(\alpha \leftarrow \beta)} \mathcal{E}(\beta, t) \rightarrow \mathcal{E}(\alpha, t),$$

where $\forall_{x \in var(r)}$ *is stands for* $\forall x_1 \forall x_2 \ldots \forall x_n$ *if* $var(r) = \{x_1, \ldots, x_n\}$ *and for* x_i *it holds that* $\mathfrak{r}(s(x_i)) = \mathcal{D}$*.*

As before, the (HT) answer streams of a program are defined as the equilibrium models.

As already mentioned in the introduction, the LARS_{HT} semantics and the standard semantics do not align in general.

Example 20 (Semantic Difference)**.** *Consider the single rule program*

$$\Pi = \{a \leftarrow \neg\neg a\}.$$

There are two possible streams which may be answer sets over the data stream $D = (\{1\}, \{1 \mapsto \emptyset\})$*, namely* $S_1 = (\{\{1\}, \{1 \mapsto \emptyset\})$ *and* $S_2 = (\{1\}, \{1 \mapsto \{a\}\})$*. The first is clearly both an answer stream and its associated interpretation is an equilibrium model at time* $t = 1$*, since it satisfies the rule and there is no substream of* S_1 *that is an interpretation stream of* D*.*

On the other hand, for the second stream this is not so clear. Under the standard semantics, we need to evaluate the FLP-reduct, resulting in

$$\Pi^{\mathcal{S}_2,1} = \Pi,$$

since $\neg\neg a$ *is satisfied. However, since* S_1 *is a substream of* S_2 *that satisfies the original program and, therefore, also the reduct, this means that* S_2 *is not an answer stream according to the standard semantics.*

Let $\mathcal{I}_i$ *be the interpretation associated to* S_i*. Then* $(\mathcal{I}_1, \mathcal{I}_2, \mathcal{I}_1, \mathcal{I}_2, H)$ *does not satisfy the formula*

$$\mathcal{E}(\neg\neg a \to a, 1) = \neg\neg a(1) \to a(1),$$

since the truth of $\neg\neg a(1)$ *at* H *is equal to the conjunction of the negations of* $\neg a(1)$ *at* H *and at* T*, which both evaluate to false. Thus,* $\neg\neg a(1)$ *is true at* H*, which implies that also* $a(1)$ *must be true at* H*. This is however not the case, since* $S_1 = (\{1\}, \{1 \mapsto \emptyset\})$*.*

There are fragments of LARS where the HT semantics and the standard semantics align. We do not go into detail here but refer the interested reader to [BDE16], where such a fragment is given. In the following, we will not restrict ourselves to such a fragment but use the HT semantics regardless. This way, we obtain the semantics of the fragment as a special case but allow more general LARS formulas, whose semantics may be different from the standard semantics, allowing the definition of a semantics for the general logic. Arguments for whether the standard semantics or the HT semantics should be preferred can be found on both sides. We stick to the HT semantics simply for the sake of simplicity, as it nevertheless illustrates the power of our approach.

Another problem that needs to be handled is that when general window functions are allowed, we lose *persistence*, which is generally expected to hold in a Here-and-There logic[1]. Persistence, says that if a formula holds at H, then it also holds at T.

Example 21. *A counterexample for persistence can be constructed using the following window function* $pick_2(S, t)$*, over the nullary predicates* $\{a, b, c\}$ *(i.e., predicates without inputs), which picks the first two atoms that hold in lexicographical order at time* t *and restricts the time interval to* $\{t\}$*.*

Consider the streams $S^H = (\{1\}, \{1 \mapsto \{a, c\}\})$ *and* $S^T = (\{1\}, \{1 \mapsto \{a, b, c\}\})$*. Since* S^H *is a substream of* S^T *for the associated interpretations* $\mathcal{I}^H, \mathcal{I}^T$ *it holds that*

[1] Additionally, one may argue that the semantics is not well defined, since the subset relation between $\mathcal{I}^H$ and $\mathcal{I}^T$ is not preserved

$(\mathcal{I}^H, \mathcal{I}^T, \mathcal{I}^H, \mathcal{I}^T, w)$ *is a pointed timed interpretation. However, while*

$$(\mathcal{I}^H, \mathcal{I}^T, \mathcal{I}^H, \mathcal{I}^T, H) \models \boxplus^{pick_2,1} c$$

holds,

$$(\mathcal{I}^H, \mathcal{I}^T, \mathcal{I}^H, \mathcal{I}^T, T) \models \boxplus^{pick_2,1} c$$

does not hold, which contradicts persistence. This is the case since $pick_2(S^H, 1) = (\{1\}, \{1 \mapsto \{a, c\}\})$ *but* $pick_2(S^T, 1) = (\{1\}, \{1 \mapsto \{a, b\}\})$.

We could restore persistence by changing the definition of the HT logic, however, the resulting behavior would be unintuitive regardless. Instead, we choose to restrict ourselves to monotone window functions, as in [BDE16].

Definition 54 (Monotonicity)**.** *We call a window function* ϖ monotone*, if for all streams* $S_1 = (T_1, v_1)$ *and* $S_2 = (T_2, v_2)$ *it holds that*

$$S_1 \subseteq S_2 \text{ implies } \varpi(S_1, t) \subseteq \varpi(S_2, t) \text{ for all } t \in T_1,$$

i.e., ϖ *preserves substreams. If* $T_1 = T_2$ *, this extends to time intervals, i.e.,* $\varpi(S_i, t) = (T_i', v_i')$ *implies* $T_1' = T_2'$ *for all* $t \in T_1$.

If we only use monotone window functions, persistence is maintained.

Lemma 55 (Persistence)**.** *Let* ϕ *be a timed* σ*-sentence that only uses monotone window functions and* $(\mathcal{I}^{H*}, \mathcal{I}^{T*}, \mathcal{I}^H, \mathcal{I}^T)$ *be a timed* σ*-HT-interpretation. Then it holds that*

$$(\mathcal{I}^{H*}, \mathcal{I}^{T*}, \mathcal{I}^H, \mathcal{I}^T, H) \models \phi \text{ implies } (\mathcal{I}^{H*}, \mathcal{I}^{T*}, \mathcal{I}^H, \mathcal{I}^T, T) \models \phi$$

Proof. The proof proceeds by structural induction on the formula ϕ. The result is well-known, except for the cases where $\phi = \boxplus^{\varpi,t}\psi$ and $\phi = \triangleright\psi$. The latter case is clear.

For the case $\phi = \boxplus^{\varpi,t}\psi$ we need to consider whether

$$(\mathcal{I}^{H*}, \mathcal{I}^{T*}, \varpi(\mathcal{I}^H, t), \varpi(\mathcal{I}^T, t), H) \models \psi \text{ implies } (\mathcal{I}^{H*}, \mathcal{I}^{T*}, \varpi(\mathcal{I}^H, t), \varpi(\mathcal{I}^T, t), T) \models \psi.$$

This also immediately follows from the induction hypothesis, since ϖ is monotone. □

This concludes the gathering of the prerequisites for a general framework for temporal and quantitative reasoning under answer set semantics, which we introduce next.

2.4.2 Algebraic LARS

In the following, we introduce algebraic LARS, a general framework capable of handling quantitative reasoning both over the set of models as well as during the determinations of models over a temporal domain.

As a basis, we use, on the one hand, LARS_{HT} restricted to monotone window functions, for the logical specification and weighted LARS_{HT} for the quantitative specification. Thus, we proceed by first introducing weighted LARS_{HT} and finally algebraic LARS as algebraic LARS measures over LARS programs with algebraic constraints and weighted LARS_{HT} formulas.

Definition 56 (Timed Syntax)**.** *For a timed signature $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$, the* weighted *timed σ-formulas over the semiring $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ are of the form*

$$\alpha ::= k \mid x \mid \phi \mid \alpha \to_{\mathcal{R}} \alpha \mid \alpha + \alpha \mid \alpha * \alpha \mid -\alpha \mid \alpha^{\mathbf{-1}} \mid \Sigma y \alpha \mid \Pi y \alpha \mid \boxplus^{\varpi,t} \alpha \mid \triangleright \alpha,$$

where $k \in R$, $x, y \in \mathcal{X}$ s.t. $\mathfrak{r}(s(x)) \subseteq R$ (i.e., x takes only values from R, whereas y may be a variable that takes values from the whole domain), $t \in \mathbb{N}$ or $s(t) = \mathcal{T}$, $y \in \mathcal{X}$, ϖ is a monotone window function, and ϕ is a timed σ-formula. The use of $-$ and $^{\mathbf{-1}}$ requires that $\oplus$ and $\otimes$ are invertible; the use of Πy requires that $\otimes$ is commutative. We define $\neg_{\mathcal{R}} \alpha = \alpha \to_{\mathcal{R}} e_\oplus$. A weighted *timed σ-sentence is a variable-free weighted timed σ-formula.*

As with the unweighted version, the main difference between timed and untimed syntax is the inclusion of the interpretation modifying operators $\boxplus^{\varpi,t}$ and $\triangleright$. Apart from this, the only change is that we allow unweighted *timed* formulas ϕ.

Accordingly, we only need to extend the definition of the weighted semantics (Definition 18) by the semantics for the interpretation modifying operators:

Definition 57 (Timed Semantics)**.** *Let $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ be a timed signature. The inductive definition of the semantics of a weighted σ-sentence over semiring $\mathcal{R}$ w.r.t. $\mathcal{I}_w = (\mathcal{I}^{H*}, \mathcal{I}^{T*}, \mathcal{I}^H, \mathcal{I}^T, w)$ is given by*

$$\begin{aligned}
[\![k]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= k, \text{ for } k \in R \\
[\![-\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= -([\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w)) \\
[\![\alpha^{\mathbf{-1}}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= ([\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w))^{-1} \\
[\![\phi]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= \begin{cases} e_\otimes, & \text{if } \mathcal{I}_w \models_\sigma \phi, \\ e_\oplus, & \text{otherwise.} \end{cases} \text{, for timed } \sigma\text{-formulas } \phi \\
[\![\alpha + \beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= [\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) \oplus [\![\beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) \\
[\![\alpha * \beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= [\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) \otimes [\![\beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) \\
[\![\alpha \to_{\mathcal{R}} \beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) &= \begin{cases} e_\otimes, & \text{if } [\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_{w'}) = e_\oplus \text{ or } [\![\beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_{w'}) \neq e_\oplus \text{ for all } w' \geq w, \\ e_\oplus, & \text{otherwise.} \end{cases}
\end{aligned}$$

$$[\![\Sigma x\alpha(x)]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) = \begin{cases} \bigoplus_{\xi\in \mathrm{supp}_\oplus(\alpha(x),\mathcal{I}_w)} [\![\alpha(\xi)]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w), & \text{if } \mathrm{supp}_\oplus(\alpha(x),\mathcal{I}_w) \text{ is finite,} \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

$$[\![\Pi x\alpha(x)]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w) = \begin{cases} \bigotimes_{\xi\in \mathrm{supp}_\otimes(\alpha(x),\mathcal{I}_w)} [\![\alpha(\xi)]\!]^\sigma_{\mathcal{R}}(\mathcal{I}_w), & \text{if } \mathrm{supp}_\otimes(\alpha(x),\mathcal{I}_w) \text{ is finite,} \\ e_\oplus, & \text{if } \mathfrak{r}(s(x)) \setminus \mathrm{supp}_\oplus(\alpha(x),\mathcal{I}_w) \neq \emptyset, \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

and extended to weighted timed σ*-sentences via*

$$[\![\boxplus^{\varpi,t}\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}^{H*},\mathcal{I}^{T*},\mathcal{I}^H,\mathcal{I}^T,w) = [\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}^{H*},\mathcal{I}^{T*},\varpi(\mathcal{I}^H,t),\varpi(\mathcal{I}^T,t),w)$$
$$[\![\triangleright\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}^{H*},\mathcal{I}^{T*},\mathcal{I}^H,\mathcal{I}^T,w) = [\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}^{H*},\mathcal{I}^{T*},\mathcal{I}^{H*},\mathcal{I}^{T*},w).$$

Again note that the semantics is the same as in Definition 18 except for the addition of the interpretation modifying operators.

We define weighted LARS formulas by extending weighted timed σ-formulas with temporal operators and assigning them a semantics by an embedding into weighted timed σ-formulas.

Definition 58 (Weighted LARS Syntax)**.** *For a timed signature* $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r}\rangle$*, the* weighted *LARS formulas over the semiring* $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ *are of the form*

$$\begin{aligned}\alpha ::= k \mid x \mid \phi \mid \alpha \to_{\mathcal{R}} \alpha \mid \alpha + \alpha \mid \alpha * \alpha \mid -\alpha \mid \alpha^{-1} \mid \Sigma y\alpha \mid \Pi y\alpha \\ \mid @_t\alpha \mid \Diamond\alpha \mid \Box\alpha \mid \boxplus^{\varpi,t}\alpha \mid \triangleright\alpha,\end{aligned}$$

where $k \in R$*,* $x, y \in \mathcal{X}$ *s.t.* $\mathfrak{r}(s(x)) \subseteq R$ *(i.e.,* x *takes only values from* R*, whereas* y *may be a variable that takes values from the whole domain),* $t \in \mathbb{N}$ *or* $s(t) = \mathcal{T}$*,* $y \in \mathcal{X}$*,* ϖ *is a monotone window function, and* ϕ *is a timed* σ*-formula. The use of* $-$ *and* $^{-1}$ *requires that* $\oplus$ *and* $\otimes$ *are invertible; the use of* Πy *requires that* $\otimes$ *is commutative. We define* $\neg_{\mathcal{R}}\alpha = \alpha \to_{\mathcal{R}} e_\oplus$*.*

The embedding of LARS formulas is extended to weighted LARS formulas as follows.

Definition 59 (Weighted Embedding)**.** *We define the weighted embedding* $\mathcal{E}^w$ *of a weighted LARS formula* α *and a time point* $t \in \mathbb{N}$ *or time variable* $t \in \mathcal{X}$*, s.t.* $s(t) = \mathcal{T}$*, via structural induction on the formula.*

$$\begin{aligned}\mathcal{E}^w(k,t) &= k \\ \mathcal{E}^w(x,t) &= x \\ \mathcal{E}^w(\phi,t) &= \mathcal{E}(\phi,t) \\ \mathcal{E}^w(\alpha \to_{\mathcal{R}} \beta,t) &= \mathcal{E}^w(\alpha,t) \to_{\mathcal{R}} \mathcal{E}^w(\beta,t) \\ \mathcal{E}^w(\alpha + \beta,t) &= \mathcal{E}^w(\alpha,t) + \mathcal{E}^w(\beta,t) \\ \mathcal{E}^w(\alpha * \beta,t) &= \mathcal{E}^w(\alpha,t) * \mathcal{E}^w(\beta,t) \\ \mathcal{E}^w(-\alpha,t) &= -\mathcal{E}^w(\alpha,t)\end{aligned}$$

$$\begin{aligned}
\mathcal{E}^w(\alpha^{-1}, t) &= (\mathcal{E}^w(\alpha, t))^{-1} \\
\mathcal{E}^w(\Sigma y \alpha, t) &= \Sigma y \mathcal{E}^w(\alpha, t) \\
\mathcal{E}^w(\Pi y \alpha, t) &= \Pi y \mathcal{E}^w(\alpha, t) \\
\mathcal{E}^w(\Diamond \alpha, t) &= \Sigma t' T(t') * \mathcal{E}^w(\alpha, t') \\
\mathcal{E}^w(\Box \alpha, t) &= \Pi t' \neg T(t') + T(t') * \mathcal{E}^w(\alpha, t') \\
\mathcal{E}^w(@_{t^*} \alpha, t) &= T(t^*) * \mathcal{E}^w(\alpha, t^*) \\
\mathcal{E}^w(\boxplus^{\varpi} \alpha, t) &= \boxplus^{\varpi, t} \mathcal{E}^w(\alpha, t) \\
\mathcal{E}^w(\triangleright \alpha, t) &= \triangleright \mathcal{E}^w(\alpha, t)
\end{aligned}$$

Here, t' is a new variable with $s(t') = \mathcal{T}$.

The rest of the definitions of programs with algebraic constraints are kept as they are in Section 2.3, except that we now require signatures, interpretations, formulas and weighted formulas to be timed. For algebraic constraints $k \sim_{\mathcal{R}} \alpha$ over weighted LARS formulas α, we take the semantics of algebraic constraints over timed formulas and replace α by the embedded formula $\mathcal{E}^w(\alpha, t)$.

With this in mind, we define LARS $\mathcal{AC}$-rules and $\mathcal{AC}$-programs.

Definition 60 ($\mathcal{AC}$-LARS Rule, $\mathcal{AC}$-LARS Program)**.** *Let $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ be a timed semiring signature over semiring $\mathcal{R}_1, \ldots, \mathcal{R}_k$. An* $\mathcal{AC}$ LARS program *is a set of* $\mathcal{AC}$-LARS rules *of the form*

$$r = H(r) \leftarrow B(r) = \phi \leftarrow \psi_1, \ldots, \psi_n, \neg\theta_1, \ldots, \neg\theta_m,$$

where each ϕ, ψ_i, θ_j is either a LARS formula, whose embedding is a timed σ-formula, or an algebraic constraint, whose weighted LARS formula's embedding is a weighted timed σ-formula. We require for each variable x occurring in r that $i \in s(x)$ iff x occurs in place of a value from the semiring $\mathcal{R}_i$. Recall here that s assigns a variable its sort and for a semiring signature $\mathfrak{r}(s(x)) \subseteq R_i$ if $i \in s(x)$.

The semantics of an $\mathcal{AC}$-LARS program with respect to a pointed LARS$_{HT}$-interpretation $(S^{H*}, S^H, S^{T*}, S^T, w, t)$ is that of the program obtained by replacing every (resp. weighted) LARS formula α by its (resp. weighted) embedding $\mathcal{E}(\alpha, t)$ (resp. $\mathcal{E}^w(\alpha, t)$), when evaluated under the pointed timed HT-interpretation $(\mathcal{I}(S^{H*}), \mathcal{I}(S^H), \mathcal{I}(S^{T*}), \mathcal{I}(S^T), w)$. Answer streams are defined as previously.

The combination with measures is now rather easy.

Definition 61 (Algebraic $\mathcal{AC}$-LARS Measure)**.** *Let $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X}, \mathcal{S}, \mathfrak{r} \rangle$ be a timed semiring signature over semirings $\mathcal{R}_1, \ldots, \mathcal{R}_k$. An* algebraic $\mathcal{AC}$-LARS measure *over σ is a tuple $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$, where*

- *Π is an $\mathcal{AC}$-LARS program over σ,*

- *$\alpha(t)$ is a weighted LARS formula whose only free variable is t over the semiring $\mathcal{R}$, whose embedding is a timed weighted σ-formula, and*
- *$\mathcal{R}$ is a semiring from $\mathcal{R}_1, \ldots, \mathcal{R}_k$.*

Let D be a data stream and let t be a time point. Then the weight of an interpretation stream *S of D is*

$$\mu(S, D, t) = \begin{cases} [\![\mathcal{E}^w(\alpha, t)]\!]_{\mathcal{R}}(S, S, S, S, H), & \text{if } S \in \mathcal{AS}(\Pi, D, t), \\ e_{\oplus}, & \text{otherwise.} \end{cases}$$

Furthermore, let $p(\overline{x})$ be a variable-free atom and suppose $\mathcal{AS}(\Pi, D, t)$ is finite, then the result of the query *$\mu(p(\overline{x}), D, t)$ is given by*

$$\mu(p(\overline{x}), D, t) = \bigoplus_{S \in \mathcal{AS}(\Pi, D, t), (S,t) \models p(\overline{x})} \mu(S, D, t).$$

Last but not least, if $\mathcal{AS}(\Pi, D, t)$ is finite, then the overall weight *$\mu(\Pi, D, t)$ is given by*

$$\mu(\Pi, D, t) = \bigoplus_{S \in \mathcal{AS}(\Pi, D, t)} \mu(S, D, t).$$

This concludes the introduction of algebraic LARS, our general framework for quantitative reasoning over a temporal domain.

Clearly, since LARS is an extension of non-ground ASP, we inherit both the previous quantitative reasoning capabilities over the set of solutions that algebraic measures over answer set programs had, as well as the quantitative reasoning capabilities used to succinctly specify the set of solutions that we introduce by adding algebraic constraints to answer set programs. Since algebraic LARS not only combines algebraic measures and constraints but also adds and more importantly integrates the reasoning capabilities of LARS in the temporal setting, we argue that algebraic LARS as a framework is uniform and general enough. Thus, we have established our initial goal of finding a general framework that combines expressive quantitative and temporal reasoning.

Example 22 (Metro Connections)**.** *We give a final version of the metro connections example, where we make use of all the components of algebraic LARS to finally specify a program that allows us to answer the initial question of how likely it is to arrive at a store given that one leaves now and takes a specified series of metro connections.*

We arrive at the algebraic $\mathcal{AC}$-LARS measure $\mu_{fin} = \langle \Pi_{fin}, \alpha_{fin}, \mathcal{P} \rangle$, where $\mathcal{P}$ is the probability semiring, Π_{fin} consists of the rules given in Figure 2.2. Here, we assume that the following predicates are extensional and, thus, given in the data stream D:

- *start(s), indicating the starting station s,*
- *goal(g), indicating the goal station g,*

$$@_1\text{location}(S) \leftarrow \text{start}(S) \tag{2.38}$$

$$@_T\text{location}(S) \leftarrow @_{T'}\text{location}(S), 0 =_{\mathbb{N}} @_{T'}\text{enter}(Id), T' =_{\mathbb{N}} T - 1 \tag{2.39}$$

$$@_T\phi_{depart} \leftarrow @_T\text{scheduled}(Id, S), \text{ where}$$

$$\phi_{depart} = \boxplus^{\text{around}_3} \Diamond\text{depart}(Id, S) \tag{2.40}$$

$$@_T\text{location}(S) \leftarrow @_T\text{exit}(Id), @_T\text{depart}(Id, S) \tag{2.41}$$

$$1 =_{\mathbb{N}} @_T \boxplus^{future} \Diamond\alpha_{enter} \leftarrow @_T\text{location}(S), \text{not goal}(S), \text{ where}$$

$$\alpha_{enter} = \neg\neg\text{depart}(Id, S) * \neg \triangleright \boxplus^{past}\Diamond\text{exit}(Id)$$

$$*((\text{depart}(Id, S) * \neg \triangleright \boxplus^{past}\Diamond\text{exit}(Id)) \rightarrow \text{enter}(Id)) \tag{2.42}$$

$$@_T\text{on\_metro}(Id) \leftarrow @_T\text{enter}(Id) \tag{2.43}$$

$$@_T\text{on\_metro}(Id) \leftarrow @_{T'}\text{on\_metro}(Id), \text{not } @_T\text{exit}(Id), T' =_{\mathbb{N}} T - 1 \tag{2.44}$$

$$\{@_T\text{exit}(Id)\} \leftarrow @_T\text{on\_metro}(Id), @_T\text{depart}(Id, S) \tag{2.45}$$

$$\leftarrow \text{planned}(M1, I1), \text{planned}(M2, I2), I2 =_{\mathbb{N}} I1 + 1,$$
$$\text{id\_of}(M1, Id1), \text{id\_of}(M2, Id2),$$
$$T_1 =_{\mathcal{R}_{\max,+}} T * @_T metro(Id1),$$
$$T_2 =_{\mathcal{R}_{\min,+}} T' * @_{T'} metro(Id2), T_1 <_{\mathbb{N}} T_2 \tag{2.46}$$

$$\leftarrow \Diamond\text{metro}(Id1), \text{id\_of}(M, Id1),$$
$$@_{T1}\text{enter}(Id1), @_{T1}\text{location}(S),$$
$$1 <_{\mathbb{N}} \text{id\_of}(M, Id2) * \Diamond(\text{location}(S) * \text{depart}(Id2, S)) \tag{2.47}$$

$$\leftarrow \text{goal}(S), \text{not} \Diamond\text{location}(S) \tag{2.48}$$

Figure 2.2: Rules of the final program for the metro connections example.

- *id_of*(m, id)*, indicating that the train with id id is driving the route of metro line* m*,*
- *planned*(m, idx)*, indicating that we plan to take the metro* m *as the* idx*-th connection, and*
- *scheduled*(id, s)*, indicating that the train with id id is scheduled at station* s

Specifically, the predicates start, goal, id_of and planned are static and only need to be given for time point 1*, whereas the predicate scheduled is dynamic, since it tells us at each time point whether a train is scheduled at a station. Therefore, it may need to be given at more than one time point.*

Furthermore, we make the following assumptions:

1. *We take each metro line at most once.*

2. *If $Id1, Id2$ are two different metro lines, then there is at most one station $S(Id1, Id2)$ such that both lines connect to it. That is, when we swap from a metro with id $Id1$ to a metro with id $Id2$, then it must happen at station $S(Id1, Id2)$.*

1. and 2. together, imply that for a given (i) sequence of metro lines that we want to take, (ii) a start station, and (iii) a goal station, there is exactly one sequence of stations that we visit. Furthermore, for each station that we visit it is uniquely determined by which metro(s) we are in, which we exit, and which we enter.

Note that the time interval T of a given data stream stream $D = (T, v)$ limits the amount of time we have to reach our goal. Alternatively, we could limit the time we have explicitly by using a window function in the last rule.

The idea behind the program Π_{fin} is that it determines from the given start station, goal station and planned metro lines, when we are in which station/metro on our way to the goal. Alternatively, the program may reject the given inputs, if a connection satisfying it is impossible. Here, the rules in Equations (2.38), (2.39) *and* (2.41) *determine that we are at a location s at a given time point, if we start there, exit a train there or are there in the previous time step and did not enter a train. Similarly, the rules in Equations* (2.43) *and* (2.44) *state that we are on a metro, if we enter it or were on it previously and did not leave it. The rules in Equations* (2.42) *and* (2.45) *tell us how we can enter and exit trains. Namely, we can enter a train if it departs at the station we are at now, which must not be the goal station. Furthermore, we must not have exited the train we enter in the past. To exit a train we must be currently on it and at a station. Additionally, the constraint in Equation* (2.47) *states that we need to enter the earliest possible train. Last but not least, the constraints in Equations* (2.46) *and* (2.48) *ensure that we take the planned route and end up at the goal station.*

The interesting part is that the rule in Equation (2.40) *encodes that scheduled trains arrive at some time point around the time point where they are scheduled rather then exactly when they are scheduled. This adds non-determinism to our program, which we want to use to assign probabilities to answer streams. In order to assign the probabilities, we use the weighted formula α_{fin}, defined as follows:*

$$\alpha_{fin} = \Box\Pi Id \Pi S \neg scheduled(Id, S) + scheduled(Id, S) * \boxplus^{around_3} \Diamond depart(Id, S) * 1/7$$

This means that for each atom $scheduled(id, s)$ at time t each of the seven time points $\{t-3, t-2, \ldots, t+3\}$ has the same probability that $depart(id, s)$ holds.

Note that more complicated probability assignments would be possible, where we do not have a uniform distribution or where we account for departure times at previous stations in the specification of probabilities. We do not go into detail here in this illustrative example, which should merely show the interplay of algebraic measures, algebraic constraints and LARS.

Overall, given a data steam D that specifies the start, goal, the time interval we have, the scheduled departures, and the planned route, we obtain the probability of reaching the start from the goal as the result of the overall weight query $\mu_{fin}(\Pi_{fin}, D, 1)$.

2.4.3 Relation to Weighted MSO and Automata

We have established that algebraic measures over ASP are expressive enough to capture many previous quantitative extensions of answer set programming. Since we can embed LARS $\mathcal{AC}$-programs into $\mathcal{AC}$-programs in a mostly straightforward manner, we also do not expect additional surprising changes in the expressivity here. However, we have yet to consider the impact of adding a temporal domain to algebraic measures on the expressivity. For the propositional fragment of LARS with time-based windows and no weights, we know that its expressive power is exactly that of finite state automata [BDE18] and, thus, also that of Monadic Second-Order Logic (MSO) due to the Büchi-Elgot-Trakhtenbrot Theorem, which states equivalent expressive power of finite state automata and MSO.

Weighted (finite state) automata generalize finite state automata, just like the weighted semantics for formulas generalizes the boolean semantics. Instead of just accepting words like finite state automata, weighted automata associate a weight over a semiring to words. A generalization of the Büchi-Elgot-Trakhtenbrot Theorem due to [DG07] states the equivalence of the expressiveness of weighted automata and a fragment of weighted MSO. It suggests itself to consider whether this result can also be lifted on the LARS side.

Weighted automata define a function from words over a finite alphabet to values in a semiring as follows:

Definition 62 (Weighted Automaton [DG07])**.** *A* weighted automaton $\mathcal{A}$ *over a finite alphabet* A *and a semiring* $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ *is a quadruple* $\langle Q, \lambda, \delta, \gamma \rangle$*, where*

(i) Q *is a finite set of states,*

(ii) $\lambda, \gamma : Q \to R$ *assign each state their respective initial and final weight, and*

(iii) $\delta : A^* \to R^{Q \times Q}$ *is a monoid homomorphism between* $(A^*, \odot, \varepsilon)$ *and* $(R, \otimes, e_\otimes)$*, where* $\odot$ *is concatenation and* ε *is the empty word.*

Here, $R^{Q \times Q}$ *denotes the set* $\{(r_{(q_1,q_2)})(q_1, q_2) \in Q \times Q \mid r_{(q_1,q_2)} \in R\}$*, which contains families of values from* R *indexed by elements from* $Q \times Q$*.*

Its behavior *is defined as*

$$\|\mathcal{A}\| : A^* \to R, w \mapsto \bigoplus_{q,q' \in Q} \lambda(q) \otimes \delta(w)_{q,q'} \otimes \gamma(q').$$

Regarding the expressive power of LARS measures, we identify here a fragment that can express the same functions as weighted automata. To this aim, we do not consider general window functions, but only those definable in monadic second-order logic (MSO). Furthermore, we consider only variable-free LARS measures that do not contain algebraic constraints.

Definition 63. *A window function ϖ is MSO definable, if there exist MSO formulas $\phi_\varpi(P, P', x), \psi_\varpi(x_s, x_e, x'_s, x'_e, x)$ s.t. for every stream $S = (T, v)$, $\varpi((T, v), t) = (T', v')$ and $t_s = \min T, t_e = \max T, t'_s = \min T', t'_e = \max T'$:*

$$\phi_\varpi(P[v], \sigma(P'), t) \text{ holds iff } \sigma(P') = P[v']$$
$$\psi_\varpi(t_s, t_e, \sigma(x'_s), \sigma(x'_e), t) \text{ holds iff } \sigma(x'_s) = t'_s, \sigma(x'_e) = t'_e$$

where for a valuation v, $P[v]$ is the vector of monadic predicates $P_a[v]$ for $a \in \mathbf{HB}$, s.t. $\forall t' \in T' : P_a[v](t') \iff a \in v(t')$.

For example, the window functions around$_n$, future and past defined in Example 18 are all MSO expressible.

Furthermore, we restrict negation $\neg$ to only occur in front of atoms. Since our proofs rely on the commutativity of multiplication we only consider commutative semirings in this section.

In the following, we show that the expressivity of weighted automata is equivalent to that of a restricted class of LARS measures.

Theorem 64 (Reduction of Weighted Automata to LARS Measures)**.** *Given a weighted automaton $\mathcal{A}$ over a finite alphabet A and semiring $\mathcal{R}$, there exists a LARS measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$, s.t.*

$$\forall w \in A^* : \|\mathcal{A}\|(w) = \mu(\Pi, \tau(w)),$$

where $\tau(w)$ is a translation from words w to pairs of data streams and time points, defined by

$$\tau(w) = \left(\left([0, |w|], t \mapsto \left\{ \begin{array}{ll} \{w_{t+1}\} & \text{if } t < |w| \\ \emptyset & \text{otherwise} \end{array} \right\} \right), 0 \right)$$

Proof (Sketch, see Appendix A.5 for the full proof). We can prove the claim by constructing a program Π that has as answer streams all possible paths through the weighted automaton for the given word and a weighted LARS formula α, whose semantics for a given answer stream is the weight of the path. By summing over all answer streams, one obtains then the behavior of the automaton on the given word as $\mu(\Pi, \tau(w))$, i.e., the overall weight with respect to $\tau(w)$. □

We proceed to establish the other direction of encodability, by expressing LARS measures via *restricted* weighted MSO formulas, which were shown to be equivalent to weighted automata [DG07]. In such formulas, universal quantifiers are restricted to first-order variables. Furthermore, the semantics of the quantified formula may only take finitely many values [DG07]. We define *restricted* LARS measures analogously as LARS measures $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$, where $\Box\beta$ can only be a subformula of α if β takes finitely many values.

Theorem 65 (Reduction of LARS Measures to Weighted MSO)**.** *Let $\langle \Pi, \alpha, \mathcal{R} \rangle$ be a restricted LARS measure over a semiring $\mathcal{R}$. Then there exists a restricted weighted MSO formula Ψ over $\mathcal{R}$, s.t.*

$$\forall D, t : \mu(\Pi, D, t) = [\![\Psi]\!]_{\mathcal{R}}(\sigma(D, t)),$$

where for a data stream D and time point t, $\sigma(D,t) = (\sigma_1(D,t), \sigma_2(D,t))$ s.t. σ_1 is the word that the weighted MSO formula is interpreted over and σ_2 corresponds to the assignments of free variables in Ψ. That is,

$$\begin{aligned}
\sigma_1(D,t)_i &= \{a \mid a \in v(i) \cap \mathbf{HB}(\Pi)^\epsilon\}, \\
\sigma_2(D,t)(x_0) &= t && \text{for the non-quantified first-order variable } x_0, \\
\sigma_2(D,t)(A) &= \{t \in \mathbb{N} \mid a \in v(t)\} && \text{for MSO variables } A. \\
\sigma_2((T,v),t)(T) &= T && \text{for MSO variable } T.
\end{aligned}$$

Proof (Sketch, see Appendix A.5 for the full proof). For the proof, we use a weighted MSO formula

$$\Psi = \exists \mathbf{A}. T(\Phi_\Pi)(\mathbf{A}) \wedge f(\alpha)$$

such that

- $f(\alpha)$ is a faithful translation of α to weighted MSO,
- $\exists \mathbf{A}$ is the sum over all interpretation streams S of D,
- $\Phi_\Pi(\mathbf{A})$ is a MSO formula, which is satisfied by $\mathbf{A}$ iff $\mathbf{A}$ corresponds to an answer stream. It can be obtained by extending the result in [BDE18] that, given a LARS program Π, one can construct a MSO formula that is satisfiable iff Π has an answer stream.
- $T(\ .\ )$ is a translation to weighted MSO that preserves the boolean semantics, due to [DG07].

By putting these things together we obtain the complete formula, which we can show to have the same semantics. □

Droste and Gastin [DG07] showed that restricted weighted MSO and weighted automata are equally expressive; hence, it follows that also restricted LARS measures and weighted automata are equally expressive. Notably, restrictedness is needed: the value $2^{|T|^2}$ of the formula $\alpha = \Box\Box 2$ over $\mathbb{N}$ is inexpressible by weighted automata [DG07].

2.4.4 Computation and Complexity

Apart from the expressivity of algebraic LARS, we are also interested in its computational complexity. The full non-ground setting with algebraic constraints is already undecidable when it comes to settling the existence of an answer stream due to Theorem 38 even without temporal operators. We thus restrict ourselves on reasoning tasks over algebraic LARS measures $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$, where Π is a variable-free LARS program without algebraic constraints and α is also variable-free.

In the light of our previous expressivity results, we first consider evaluating weighted LARS formulas, by encoding them as weighted automata. This seems promising, as weighted automata have polynomial data complexity when counting the arithmetic operations. Unfortunately, any encoding is non-elementary in general.

Theorem 66. *For every constant k, there is no translation of restricted weighted LARS formulas of size n to weighted automata s.t. the size of the automaton can be bounded by a function of the form $\exp(k)$, where $\exp(0) = n, \exp(k+1) = 2^{\exp(k)}$.*

Proof. Towards a contradiction, assume such a translation τ exists.

Given an unweighted MSO formula ϕ over words in some alphabet A, we can consider the MSO definable window function ϖ_ϕ, given by

$$\varpi_\phi(S,t) = \begin{cases} S & \text{if } \models \phi(\sigma(S,t)), \\ (\emptyset,\emptyset) & \text{otherwise.} \end{cases}$$

and the formula

$$\alpha = \boxplus^{\varpi_\phi} \Diamond \top.$$

Then α is a restricted weighted LARS formula over $\mathbb{B}$ and for the weighted automaton $\mathcal{A}$, which corresponds to the LARS measure α it holds that

$$\forall w \in A^* : \|\mathcal{A}\|((\{w_1\}, \ldots, \{w_{|n|}\})) = \top \iff \models \phi(w).$$

Note that the automaton is over the alphabet $\mathcal{P}(A)$, however we only use the singleton letters $\{a\}$ for $a \in A$. We can simply reduce the automaton to the alphabet A. The specification of the new automaton is then single exponentially smaller. We can interpret it as a non-deterministic finite state automaton and construct a deterministic version of it. This version is only single exponentially larger than the non-deterministic one. This is however a contradiction, since this entails the existence of a translation from MSO formulas to finite state automata which can be bounded using a tower of exponentials with finite height. □

We see that the non-elementary complexity already follows from Meyer and Stockmeyer's result [SM73] that translating MSO to finite automata has non-elementary complexity.

However, if we consider windows common in practice like time-based windows and restrict rules to fragments like plain LARS [BDE18], we can obtain a translation from restricted LARS measures to weighted automata that can be bounded double exponentially in size.

Alternatively, we can evaluate weighted LARS formulas using a Turing machine. For this we consider as in [BDE18] window functions that can be evaluated in polynomial time. We formally define the evaluation problem as follows:

- **EVAL-wLARS**: given a variable-free weighted LARS formula α over a semiring $\mathcal{R}$, a stream S, and a timepoint t in S, compute the value of $[\![\alpha]\!]_{\mathcal{R}}(S,t)$.

The complexity of this problem may grow arbitrarily depending on the semiring and the problem can be undecidable in general (unless $\mathcal{R}$ is explicitly represented and functions for the arithmetic operations are provided in the input). Beck, Dao-Tran, and Eiter [BDE18] showed that evaluating LARS formulas is PSPACE-complete; thus PSPACE-hardness is a lower bound for any non-trivial semiring. In fact, there is even a semiring such that EVAL-wLARS is FPSpace(poly)-complete, where FPSpace(poly) contains the functions with polynomial output size computable by Turing machines in polynomial space.

Lemma 67 (FPSpace(poly)-hardness)**.** *Let $\mathcal{Q} = (2^{\mathbb{N}}, \min, \cup, \mathbb{N}, \emptyset)$, where the minimum is taken w.r.t. to the order $\succ$, where $X \succ Y$ holds iff*

$$\exists n \in X : (\forall n' < n : n' \in X \iff n' \in Y) \wedge n \in X \setminus Y.$$

That is, $X \succ Y$ iff the smallest number that X and Y disagree on is in X.

*The problem **EVAL-wLARS** over $\mathcal{Q}$ is FPSpace(poly)-hard.*

Proof (Sketch, see Appendix A.6 for the full proof). We reduce the problem QBF-sat-search of finding a satisfying assignment for the free variables of a QBF formula ϕ, given that it is satisfiable to **EVAL-wLARS** over $\mathcal{Q}$. Since it is known that QBF-sat-search is FPSpace(poly)-complete [HSH12], this will establish the result.

Here, we use the idea of Beck, Dao-Tran, and Eiter [BDE18] who introduced window functions of the form $set : z$ for a propositional variable, which either remove or leave z in the interpretation depending on the time point. This together with the temporal operators $\Diamond$ and $\Box$ is enough to model quantifiers. The semiring $\mathcal{Q}$ then only gathers the assignment. □

We see that we cannot guarantee efficient evaluation in general. However, in the following we give restrictions on formulas and semirings that allow for efficient evaluation.

Efficiently Evaluable Fragments

The inefficiency of the evaluation of a weighted formula α under some stream is due to the unrestricted use of quantifiers, i.e. $\Box, \Diamond$ in α. If we bound the nesting depth of quantifiers, denoted $\text{qdepth}(\alpha)$, by some constant k, we obtain that under reasonable restrictions on the semiring, the time needed for evaluation of a formula given some stream is polynomial in the size of the formula and the stream. The reasonable restriction is the previously defined efficient encodedness. Recall:

Definition 36 (Encoding Function, Efficiently Encoded Semiring)**.** *Let* $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ *be a semiring. Then an injective function* $e : R \to \{0,1\}^*$ *is an* encoding function*.*

Given an encoded value $e(r)$ *we define* $\|r\|_e$*, the* size of r w.r.t. e*, as the length of the bitstring* $e(r)$*, i.e.* $|e(r)|$*.*

Let $\mathcal{R}$ *be a semiring and* $e : R \to \{0,1\}^*$ *an encoding function. Then* $\mathcal{R}$ *is* efficiently encoded *by* e*, if there exists a polynomial* $p(x)$ *s.t. for all* $e(r_1), \ldots, e(r_n) \in e(R)$ *it holds that*

1. $\|\bigotimes_{i=1}^n r_i\|_e \leq p(n) \max_{i=1,\ldots,n} \|r_i\|_e$,
2. $\|\bigoplus_{i=1}^n r_i\|_e \leq p(\log_2(n)) \max_{i=1,\ldots,n} \|r_i\|_e$,
3. $\max(\|-r\|_e, \|r^{-1}\|_e) \leq \|r\|_e + p(0)$,
4. $e(r), e(r') \mapsto e(r \odot r')$ *is in FP for* $\odot = \oplus, \otimes$*, and*
5. $e(r) \mapsto e(f(r))$ *is in FP for* $f(.) = -(.), (.)^{-1}$.

Conditions 1) and 2) ensure that successive multiplications resp. additions do not cause space explosion, even for sums with exponentially many terms. Similarly, 3) ensures that inverting a value with respect to addition or multiplication may only lead to a constant space increase. Conditions 4) and 5) are necessary since we at least need single operations to be tractable if we want to solve problems over a semiring efficiently. The idea behind these conditions is to separate encodings that behave "efficiently" both with respect to space and time and those that do not. For this, we use restrictions that mirror and slightly relax the properties that the prototypical binary encoding of integers satisfies.

The restriction on the encoding function is mild in practice, since most practically used semirings, like $\mathbb{Q}, \mathbb{N}, \mathbb{B}, \mathcal{P}(A), \mathcal{R}_{\max,+}$ satisfy it.

Theorem 68 (Complexity of Evaluation I)**.** *Let* $\mathcal{R}$ *be a fixed semiring that is efficiently encoded by* e*, and let* $k \in \mathbb{N}$ *be a fixed constant. Then for any weighted LARS formula* α *over* $\mathcal{R}$ *s.t.* $qdepth(\alpha) \leq k$*, we can calculate* $[\![\alpha]\!]_{\mathcal{R}}(S,t)$ *in polynomial time in the size of* α *and* S*.*

Notably, for this theorem, we could even weaken condition (2) on the addition of the semiring to be the same as condition (2) on the multiplication of the semiring.

Proof (Sketch, see Appendix A.6 for the full proof). The proof is by structural induction on the formula α, with the induction invariant that the time $t(\alpha)$ needed is in $\mathcal{O}(N^{n\cdot(k+1)})$, where N is the size of the input, $n \in \mathbb{N}$ is a constant that is not depending on the input and $k = \text{qdepth}(\alpha)$. Furthermore, the size $s(\alpha)$ of the representation of the obtained value, i.e. $\log_2(e([\![\alpha]\!]_{\mathcal{R}}(S,t)))$, is in $\mathcal{O}(N \cdot N^k)$. □

This result is as expected from LARS, where under similar restrictions one can perform model checking in polynomial time [BDE18]. However the FPSpace(poly)-membership does not seem to generalize to arbitrary formulas. If we consider the formula $\Box^k 2 = \Box\Box^{k-1}2$, which is equal to $2^{|T|^k}$ when evaluated over $\mathbb{N}$, we see that the binary representation of the result has exponential size.

Similarly, we can lift the result to algebraic queries.

Theorem 69 (Complexity of Evaluation II)**.** *Let $\mathcal{R}$ be a fixed semiring that is efficiently encoded by e which supports natural addition, and $k \in \mathbb{N}$ a fixed constant. Then for any LARS measure $\mu = \langle \Pi, \alpha, \mathcal{R}\rangle$ s.t. qdepth$(\alpha) \leq k$, stream S, data stream D and timepoint t, computing $\mu(\Pi, D, t)$ and $\mu(S, D, t)$ are both in FPSpace(poly).*

Proof. Deciding $S \in AS(\Pi, D, t)$ is feasible in PSPACE [BDE18], and Π has at most $2^{|\mathbf{HB}(\Pi)^I||T|} \leq 2^{N^2}$ answer streams S for D, where N is the size of the input. For each of them $e(\mu(S, D, t)) \in \mathcal{O}(2^{N^{k+1}})$, thus $e(\mu(\Pi, D, t)) \in \mathcal{O}(2^{N^{k+3}})$ as e supports natural addition; thus the binary representation of $e(\mu(S, D, t)$ resp. $e(\mu(\Pi, D, t))$ has size polynomial in N. Furthermore, iterating over all S to compute $e(\mu(D, t))$ is doable in polynomial space. □

The problem is FPSpace(poly)-complete in general. For restricted classes of programs Π or weighted formulas, we may end up with problems that are complete for classes contained in FPSpace(poly). For example, when $\alpha = 1$ and $\mathcal{R} = \mathbb{N}$, the evaluation of $\mu(\Pi, D, t)$ amounts to model counting of programs, which is #P-complete for normal ASP programs and #coNP-complete for disjunctive ASP programs, as follows from reducibility to results for #SAT resp. #CIRC [DHK05].

Preferential Reasoning

Given a strict partial order $>$ (or equivalently a strict preorder) defined on the set R of elements of the semiring $\mathcal{R}$, we can use any quantitative query defined by some LARS measure μ as the objective function with respect to which streams are ranked in preference based reasoning.

Definition 70 (Preferred Stream)**.** *We say an answer stream $S \in \mathcal{AS}(\Pi, D, t)$ is a* preferred stream *w.r.t. a LARS measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$, data stream D, timepoint t, and strict order $>$ on R, if*

$$\nexists S' \in \mathcal{AS}(\Pi, D, t) : \mu(S', D, t) > \mu(S, D, t).$$

Given some LARS measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$ with a strict partial order $>$ on R, we consider the following two problems:

- **Preference Checking (PC)**: Given a stream S, LARS measure μ, data stream D and time t, check whether S is preferred.
- **Brave Preferential Reasoning (BPR)**: Given a LARS measure μ, data stream D, time t and an atomic formulas a check whether there exists a preferred stream S such that $(S, t) \models a$.

We obtain the following results:

Theorem 71. *There are a semiring $\mathcal{R}$ efficiently encoded by e, $>$ a strict order on R such that $r_1 > r_2$ is decidable in polynomial time, and a fixed constant $k \in \mathbb{N}$ such that for every LARS measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$ such that $qdepth(\alpha) \leq k$, and it holds that*

(i) PC and BPR are PSPACE-complete.

If in addition every $\alpha \leftarrow \beta \in \Pi$ fulfills $\max(qdepth(\alpha), qdepth(\beta)) \leq k$, then

(ii) PC is Π_2^p-complete, and

(iii) BPR is Σ_3^p-complete.

Proof. (i) PSPACE-hardness follows from the fact that already checking whether a stream is an answer stream is PSPACE-hard for general LARS programs [BDE18]. For membership we can use the fact that we can iterate over all answer streams in PSPACE [BDE18].

(ii) Let Π some LARS program with bounded quantifier nesting. Satisfiability checking for Π is Σ_2^p-complete [BDE18]. We consider the LARS measure

$$\mu = \langle \{\alpha \leftarrow \beta \wedge a \mid \alpha \leftarrow \beta \in \Pi\} \cup \{a \leftarrow \neg b, b \leftarrow \neg a\}, b, \mathbb{B} \rangle$$

for some new variables a, b, order $\bot < \top$ and stream $S = (T, v)$ s.t. $v(t) = \{a\}$. Since this stream is an answer stream of the program used in μ we have that it is preferred if there does not exist an answer stream that asserts b. This is the case iff the original program Π does not have an answer stream.

Membership in Π_2^p can be shown, by non-deterministically guessing S' and checking whether $S' \in \mathcal{AS}(\Pi, D, t)$ and $\mu(S', D, t) > \mu(S, D, t)$.

(iii) We show Σ_3^p-hardness by a reduction of checking whether a QBF formula of the form $\Phi = \exists X : \forall Y : \exists Z : \phi(X, Y, Z)$ is true. We know that unsatisfiability checking for LARS programs Π with $\text{depth}(\Pi) \leq k$ is Π_2^p-complete. Thus, we can find a program $\Pi(X)$ with extensional atoms X, s.t. for any assignment x to the variables X it holds $\forall Y : \exists Z : \phi(x, Y, Z)$ is true iff $\Pi(x)$ is unsatisfiable at time t. Consider now

$$\begin{aligned}\Pi^* = &\{\alpha \leftarrow \beta \wedge a \mid \alpha \leftarrow \beta \in \Pi(X)\} \\ &\cup \{a \leftarrow \neg b, b \leftarrow \neg a\} \\ &\cup \bigcup_{i=1}^{n} \{X_i \leftarrow \neg \overline{X}_i, \overline{X}_i \leftarrow \neg X_i\}.\end{aligned}$$

The rules in the third line ensure a choice of the value of X_i. If for the given choice the program $\Pi(X)$ is unsatisfiable the only answer stream of Π^* with those choices for X is the one that does not contain a, essentially deactivating all the other rules of the original program. Therefore, Φ is true iff there exist choices for X s.t. every answer set of $\Pi^*(X)$ contains b. Now consider the LARS measure

$$\left\langle \Pi^*, a \wedge \{a\} \vee \bigvee_{i=1}^{n} X_i \wedge \{X_i\}, \mathcal{P}(\{a, X_1, \ldots, X_n\}) \right\rangle,$$

where intuitively, the semiring values $\{a\}, \{X_1\}, \ldots, \{X_n\}$ record when the corresponding variables are true. Let, furthermore, $>$ be the order on $\mathcal{P}(\{a, X_1, \ldots, X_n\})$ s.t.

$$A_1 > A_2 \iff a \in A_1 \wedge a \notin A_2 \wedge A_1 \setminus \{a\} = A_2.$$

Then there exists a preferred answer stream S s.t. $(S, t) \models b$ iff there is an assignment x to the variables X s.t. $\forall Y : \exists Z : \phi(x, Y, Z)$ holds.

For membership, we can guess a interpretation stream S for D which contains a and do (PC) on it. □

We remark that BPR has a lower complexity, namely Δ_3^p, when one considers only pairs $(\mathcal{R}, >)$ such that efficient binary search is possible, as for BPR in the presence of weak constraints [Leo+06].

2.4.5 Conclusion

Arguably, the definition of algebraic LARS is somewhat tedious and requires a lot of steps. However, this seems hardly avoidable, since we require non-ground programs, temporal operators, and a weighted version of Boolean logic that furthermore needs to be in line with the non-monotonicity of answer set semantics.

Regardless of its perhaps alleged lack of elegance, algebraic LARS satisfies the goals we initially stated: it is capable of expressive temporal as well as quantitative reasoning

under answer set semantics. We have seen the quantitative reasoning capabilities without the temporal domain separately in Sections 2.2 and 2.3 and shown that in this setting the algebraic approach is general enough to cover existing quantitative extension and even bring new possibilities. By lifting the definitions to the temporal setting using the genericity of weighted (Here-and-There) Logic, we however do not only gain the three different capabilities separately but also immediately their integration. That is, we can specify quantitative constraints over a temporal domain natively and answer quantitative queries over the set of answer streams by assigning each answer stream a complex weight that depends on the true atoms at different time points in the stream in a non-trivial manner. We have demonstrated this extensively in Example 22.

Having established this powerful theoretical framework, we are now left with the task of its analysis and implementation. In terms of analysis we have already established a number of results. We have safety properties and conditions for finite groundability for $\mathcal{AC}$-programs that give us decidable fragments. These properties only need to be lifted to LARS $\mathcal{AC}$-programs, which should work (we do not expect any technical difficulties) and, thus, is not discussed in this thesis. Additionally, we showed that a restricted fragment of variable-free LARS measures without algebraic constraints has the same expressivity as a restricted fragment of weighted MSO and weighted automata. This correspondence allows us to see LARS measures as a rule based specification language for weighted automata. Apart from this, we considered the complexity of common reasoning tasks. We already established many results in this direction, some of them even completeness results: for preferential reasoning over variable-free LARS programs with bounded quantifier depth, we obtained that checking preferredness and brave inference are Π_2^p- and Σ_3^p-complete, respectively, under reasonable restrictions on the semiring.

A big open question is how to properly handle semirings and what their influence on the complexity is. Efficient encodedness of semirings allows us to give some upper bounds but we are missing general lower bounds, and a more fine grained analysis of the complexity in dependence of the semiring. This is the case both for the evaluation of algebraic constraints as well as for the evaluation of algebraic measures. Since we consider these questions to be highly important both from a theoretical point of view and for a possible future implementation, we dedicate Chapter 3 to them.

In terms of complexity, it would also be interesting to see whether there are fragments of lower complexity. This would also be interesting in terms of a possible implementation, since for the general setting many of the typical reasoning tasks are undecidable. And even in the decidable cases that we already know, we have to expect rather high intractability due to our complexity results.

Towards an implementation, this chapter does not give any insights yet. However, in order to fully make use of the benefits of this work in a practical setting an implementation is desirable. As mentioned above, a full implementation requires an in depth analysis of fragments with lower complexity. Apart from this, it is naturally quite challenging to implement all the three aspects of Algebraic LARS, i.e., algebraic measures, algebraic constraints and temporal reasoning, in the same solver. Since this would go beyond the

scope of a doctoral thesis, we instead choose to focus on the arguably least explored aspect. Given that there are already efficient implementations of LARS [BEF17; EOS19] and other temporal frameworks [Cab+19], as well as support for many constructs expressible with algebraic constraints in standard solvers [Geb+14; Leo+06], this means we consider the practical evaluation of algebraic measures in more detail. Here, there are implementations for some specific use cases such as probabilistic reasoning [Fie+15; LTW17], and optimization [Geb+14]. The only implementation of the general algebraic setting available is for a restricted language fragment [KVD11]. Thus, we consider in Chapter 4 how we can efficiently evaluate algebraic measures over general semirings and normal answer set programs.

CHAPTER 3

Complexity of Counting over Semirings

3.1 Introduction

Our work to extend Answer Set Programming (ASP) with quantitative reasoning capabilities heavily relies on semirings to achieve a high level of generality in a uniform manner. The algebraic structure of the semiring allows us, on the one hand, to restrict computations to a reasonable setting, since the addition and the multiplication of the semiring must satisfy certain axioms. On the other hand, by parameterizing definitions with a semiring we become very flexible such that our definitions cover a manifold of special cases, when they are instantiated with the correct respective semirings.

Unsurprisingly, we are not the first to find that semirings are a well-suited basis for defining quantitative frameworks that uniformly capture a multitude of problems. For example, Kimmig, Van den Broeck, and De Raedt [KVD17] and Belle and De Raedt [BD20] provide an extensive list of problems that are easily formulated by instantiating a semiring framework with different semirings. To name a few, SAT corresponds to the Boolean semiring $\mathbb{B}$; #SAT corresponds to the semiring of the natural numbers $\mathbb{N}$; parameter learning corresponds to the gradient semiring [Man+21]; and expected utility computations are handled by the expectation semiring [Eis02].

Such semiring paradigms, which allow us to introduce a multitude of quantitative versions of a qualitative problem in a uniform manner, are fruitfully used in a manifold of ways, including but not limited to:

- Algebraic Measures Section 2.2, which encompass many quantitative reasoning capabilities over the set of answer sets of a logic program;

- Algebraic Model Counting (AMC) [KVD17], which generalizes Weighted Model Counting to work with semiring values as weights;
- Semiring-based Constraint Satisfaction Problems (SCSPs), which were shown to encompass Fuzzy CSPs, Probabilistic CSPs, Weighted CSPs and more, each corresponding to SCSPs over another semiring [Bis+99];
- Algebraic Constraints Section 2.3, which extend the specification language of ASP to capture and generalize many previous quantitative constructs of ASP in a uniform manner;
- Provenance Semirings [GKT07], which were shown to be a useful tool to express which-, why- and bag-why-provenance in the context of positive relational algebra queries, by employing polynomial semirings;
- Algebraic Problog [KVD11], which is not only available as an implementation for probabilistic logic programming, but also capable of parameter learning and offers most probable explanation inference by employing specialized semirings.

Of course, for us algebraic measures and algebraic constraints are the most important concepts on this list. However, as we will argue also all the other problems fall into the same class of problems computationally. As such, their computational aspects should be treated together.

This long list naturally begs the question what the computational complexity of reasoning in these frameworks is and, more importantly, how it depends on the semiring the framework is instantiated with. While it is known that we face #P-hardness when evaluating a problem over the semiring $\mathbb{N}$ of the natural numbers, and that the Boolean semiring $\mathbb{B}$ leads to instances that are in NP, this does not give us fully satisfactory insights. Indeed, these results are especially unsatisfactory in the light of Toda's Theorem [Tod89], which shows that not only NP but even the whole polynomial hierarchy PH is contained in $\mathrm{P}^{\#\mathrm{P}[1]}$, which means that each problem in PH can be solved in polynomial time with a single call to an #P oracle (tantamount to solving a single #SAT instance). This gives us the intuition that there is likely a large gap between NP and #P and, thus, also between the semirings $\mathbb{B}$ and $\mathbb{N}$.

We are thus interested in a fine grained analysis of the complexity depending on the semiring parameter. Here, according to the best of our knowledge, current complexity results for general semirings are still preliminary.[1] Completeness results are only known for a few specific semirings [SI96], NP-hardness is only known over *idempotent* semirings [Bis+99] and the general EXPTIME upper bound that follows from the results of Kimmig, Van den Broeck, and De Raedt [KVD17] assumes that multiplication and addition can be done in constant time.

[1] This does not extend to the field of fixed-parameter-tractability. There is a wide range of results for semiring frameworks in this area, cf. [Gan+22; FD16; BDP09]

In this chapter, we therefore take a closer look at the computational complexity of reasoning frameworks that depend on a semiring parameter such as the ones above. For this, we first provide in Section 3.2 some necessary preliminaries. We then provide in Section 3.3 some examples that motivate the paradigm of computations with semirings as parameter and give some guidance for its development. Next, we lay in Section 3.4 the basis of a methodical complexity analysis of semiring-based formalisms by introducing NP($\mathcal{R}$), a semiring version of NP, and SAT($\mathcal{R}$) as a canonical complete problem. Afterwards, we immediately employ in Section 3.5 the NP($\mathcal{R}$)-completeness result for SAT($\mathcal{R}$), by providing a variety of NP($\mathcal{R}$)-completeness and -hardness results for different semiring frameworks, including algebraic measures and algebraic constraints. In the subsequent Section 3.6, we explore the relation of NP($\mathcal{R}$) to classical complexity classes in the form of different upper and lower bounds. In Section 3.8, we consider related work and discuss our results, while in the final Section 3.9 we conclude with issues for future research.

3.2 Preliminaries

We start by giving the necessary preliminaries. We assume knowledge of the basics of propositional logic. In this context, we consider propositional theories T over a set $\mathcal{V}$ of Boolean variables (i.e., propositional atoms) using the Boolean connectives $\vee$, $\wedge$, and $\neg$ for disjunction, conjunction, and negation, respectively; further connectives, e.g. material implications $\rightarrow$, may be defined as usual. Furthermore, $\bot$ and $\top$ are shorthands for an unsatisfiable (false) and a tautologic (true) formula, respectively. For interpretations of T, we consider subsets $\mathcal{I}$ of $\mathcal{V}$, where $v \in \mathcal{I}$ indicates that v is satisfied and $v \notin \mathcal{I}$ indicates that $\neg v$ is satisfied. Satisfaction of formulas, theories ϕ etc. by an interpretation $\mathcal{I}$ is then defined as usual and denoted by $\mathcal{I} \models \phi$.

Furthermore, we make use of generated semirings.

Definition 72 (Generated Semiring)**.** *Let $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ be a semiring. For any $R^\star \subseteq R$, the semiring generated by $R^\star$, denoted $\langle R^\star \rangle_\mathcal{R}$, is the least (w.r.t. $\subseteq$) semiring $(R', \oplus, \otimes, e_\oplus, e_\otimes)$ s.t. $R^\star \subseteq R'$.*

Example 23. *$\mathbb{N}$, the semiring of the natural numbers, is generated by $\{1\}$ or even by the empty set. The same holds for the Boolean semiring. On the other hand, the semiring $\mathbb{Q}$ of the rational numbers needs more elements to be generated. $\{1/n \mid n \in \mathbb{N}\}$ and $\{1/p \mid p \text{ is prime}\}$ are possible generators.*

For our complexity considerations we need problem reductions. Since we are studying functional complexity, we use the following notions of reductions.

Definition 73 (Metric [Kre88], Counting [CKS01] & Karp Reduction [CKS01])**.** *Let $f_i : \Sigma^* \rightarrow \Sigma^*, i = 1, 2$ be functions, then*

- *a* metric reduction from f_1 to f_2 *is a pair* T_1, T_2 *of polynomial time computable functions* $T_1 : \Sigma^* \to \Sigma^*$, $T_2 : \Sigma^* \times \Sigma^* \to \Sigma^*$ *such that* $f_1(x) = T_2(x, f_2(T_1(x)))$ *for every* $x \in \Sigma^*$.
- *a* counting reduction *is a metric reduction such that* $T_2(x, y) = T_2(x', y)$ *for all* x, x', y.
- *a* Karp reduction *(or* parsimonious reduction*) is a metric reduction such that* $T_2(x, y) = y$ *for all* x, y.

The different reductions are differently restricted. Intuitively, we have a metric reduction from f_1 to f_2, if we can compute $f_1(x)$ using polynomial time and access to one oracle call that solves $f_2(x')$. With counting reductions we still have polynomial time and one oracle call but cannot use the original input in the reconstruction of the result for f_1 from the result for f_2. Karp reductions are even more restricted, since there we cannot perform any reduction and the result for f_1 must be the same as the one for f_2 with the modified input. Karp reductions typically allow us to prove that computing f_2 is at least as hard as computing f_1, whereas the others give us slightly weaker results.

Also, we use some well-known complexity classes.

- #P [Val79] (resp. GapP [FFK94]): the functions definable as the number of accepting paths (resp. minus the number of rejecting paths) of a nondeterministic polynomial time Turing Machine (NTM);
- OptP [Kre88]: the functions definable as the maximum output of a polynomial time NTM;
- FP: the functions computable in polynomial time;
- $\mathrm{FP}_{\|}^{\mathcal{C}}$ [JT95]: the functions computable in polynomial time with parallel queries to a $\mathcal{C}$ oracle, where $\mathcal{C}$ is a complexity class;
- FPSpace(poly) (resp. FPSpace(exp)): the functions computable in polynomial space with polynomial (resp. unrestricted and thus exponential) size output;
- $\mathcal{C}/poly$ [KL82]: the problems solvable in $\mathcal{C}$ with polynomial advice, where $\mathcal{C}$ is a complexity class, i.e., there is a function f such that we can solve the problem for all inputs x in $\mathcal{C}$ given $x, f(|x|)$, where $|x|$ denotes the length of the input and $|f(x)| \in \mathcal{O}(|x|^k)$ for constant $k \in \mathbb{N}$;
- $\mathrm{Mod}_p\mathrm{P}$ [Her90]: the languages L described as $x \in L$ iff $f(x) \not\equiv 0 \mod p$ for some f in #P;
- $\mathcal{C}_1^{\mathcal{C}_2[f(n)]}$: the problems solvable in $\mathcal{C}_1$ with at most $f(n)$ calls to a $\mathcal{C}_2$ oracle, where $\mathcal{C}_1$ and $\mathcal{C}_2$ are complexity classes. Here, $\mathcal{C}_1$ is a Turing machine based complexity class (such that an oracle makes sense);

- BPP [Gil77]: the languages L for which some NTM T exists such that if $x \in L$ at least $2/3$ of the computation paths of T on x accept and if $x \notin L$ at most $1/3$ of the computation paths of T on x accept.

The exact relationship between the different complexity classes is often unclear. However, it is known that #P, OptP, Mod_pP and NP are contained in FPSpace(poly). Apart from that, #P is considered to be at least as hard as NP and Mod_pP, as we can obtain the solution of a problem in NP or Mod_pP via one #P-oracle call. As for the relationship between NP and Mod_pP, it is known that $\text{NP} \subseteq \text{BPP}^{\text{Mod}_p\text{P}}$ [BG81]. In regards to the power of Turing machines with advice, we note that the advice function does not need to be computable and thus, P/poly contains some undecidable problems. Nevertheless, the power of advice is limited: if $\text{NP} \subseteq \text{P/poly}$ or $\#\text{P} \subseteq \text{FP/poly}$ then the polynomial hierarchy collapses to a finite level [KL80], which is considered to be unlikely. On the other hand, it is known that $\text{BPP} \subseteq \text{P/poly}$ [BG81].

Hardness and completeness are defined as usual:

Definition 74 (Hardness, Completeness)**.** *A problem P is* $\mathcal{C}$-hard *for a complexity class $\mathcal{C}$ under X-reductions, if every problem $P' \in \mathcal{C}$ can be reduced to P by some X-reduction; P is* $\mathcal{C}$-complete *under X-reductions, if in addition $P \in \mathcal{C}$.*

3.3 Semiring Paradigm

Before we introduce new complexity classes to capture the computational complexity of frameworks that depend on a semiring parameter, we first introduce two additional prominent examples of such frameworks. This, on the one hand, shows that an in depth complexity study is not only useful for us in the context of algebraic measure and algebraic constraints but that such frameworks are more widely spread. On the other hand, it provides an insight into the structure that is shared by different such frameworks. The latter is also important as it tells us more about the nature of the problems at hand and thus guides us to an appropriate definition of a machine model for computations involving semirings.

Algebraic Model Counting The first framework we consider is Algebraic Model Counting (AMC), introduced by Kimmig, Van den Broeck, and De Raedt [KVD17]. Intuitively, it is a generalization of weighted model counting for propositional formulas, with weights that can be values from a semiring.

Definition 75 (AMC)**.** *Given a propositional theory T over propositional variables $\mathcal{V}$, a commutative semiring $\mathcal{R}$, and a labeling function $\alpha : L \to R$ that maps the literals L over $\mathcal{V}$ to R,* AMC *is to compute the value*

$$A(T) = \bigoplus_{\mathcal{I},\ s.t.\ \mathcal{I} \models T} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \notin \mathcal{I}} \alpha(\neg v).$$

That is, we take a sum over all interpretations that satisfy a propositional theory T, where each addend is a product of the weights of the literals satisfied by the interpretation. Contrary to weighted model counting, where sum and product need to be the usual addition and multiplication over the reals, they can be from any semiring here. The same holds for the weights, which also do not necessarily need to be reals.

In a sense, AMC is to algebraic measures what SAT is to ASP. AMC is a special case of algebraic measure evaluation, where the weighted formula does not contain complex expressions but only a product of weights of literals. Furthermore, instead of a propositional theory, we make use of an answer set program in the context of algebraic measures.

Besides the standard applications in SAT, #SAT, and probabilistic inference, the authors showed that AMC can be used to perform sensitivity analysis of probabilistic inference w.r.t. a parameter by using the semiring of the polynomials with coefficients in $[0, 1]$. For more semirings that allow additional applications, like the construction of tractable circuit representations, we refer the reader to Kimmig, Van den Broeck, and De Raedt [KVD17].

We see that regardless of the specific semiring, some of the structure stays the same: Similarly to SAT, MaxSAT and #SAT, we guess an interpretation, obtain a value based on the truth of the literals and their relation to the propositional theory and in a last step perform some form of aggregation operation over all interpretations. In the case of SAT, this corresponds to checking whether a given assignment satisfies the theory and aggregating these values by quantifying existentially over all possible interpretations. For MaxSAT on the other hand, we associate a number with a given interpretation and aggregate these values by taking the maximum.

Semiring-based Constrained Satisfaction Problems A similar structure can be observed for Semiring-based Constrained Satisfaction Problems (SCSPs). Bistarelli, Montanari, Rossi, Schiex, Verfaillie, and Fargier [Bis+99] introduced them as a generalization of constraint satisfaction problems parameterised with *c-semirings* $\mathcal{R}$, which are idempotent commutative semirings such that the axiom $\forall r \in R : r \oplus e_\otimes = e_\otimes$ holds. This restriction is due to the fact that SCSPs were defined to capture semantics for the levels of consistency of Constraint Satisfaction Problems (CSPs), rather than semantics for general quantitative reasoning.

Definition 76 (Constraint System, Constraint Problem)**.** *A* constraint system *is a tuple* $CS = \langle \mathcal{R}, D, V \rangle$*, where* $\mathcal{R}$ *is a c-semiring,* D *is a finite domain, and* V *is an ordered set of variables. A constraint over* CS *is a pair* $\langle def, con \rangle$*, where* $con \subseteq V$ *and* $def : D^{con} \to R$*, given explicitly as a set of key-value pairs, is the* value *of the constraint. Here,* D^{con} *denotes the set* $\{(d_x)_{x \in con} \mid \forall x \in con : d_x \in D\}$*.*

A constraint problem P *over* CS *is a pair* $P = \langle C, con \rangle$*, where* C *is a multiset of constraints over* CS *and* $con \subseteq V$*.*

Using D^{con} instead of $D^{|con|}$ is helpful, since it allows us to associate inputs and their variable.

Bistarelli, Montanari, Rossi, Schiex, Verfaillie, and Fargier [Bis+99] showed that SCSPs correspond to classical CSP, probabilistic CSP, weighted CSP and fuzzy CSP when the c-semiring is respectively chosen as $\mathbb{B}, ([0,1], \max, \cdot, 0, 1), \mathcal{R}_{\min,+}$ and $([0,1], \max, \min, 0, 1)$.

Example 24. *The following are constraints over* $CS = \langle \mathcal{R}_{\min,+}, \{a,b\}, \{x,y\}\rangle$*:*

$$c_1 = \left\langle \begin{array}{l} aa \mapsto 1 \\ ab \mapsto 2 \\ ba \mapsto 3 \\ bb \mapsto 4 \end{array}, \{x,y\} \right\rangle, c_2 = \left\langle \begin{array}{l} a \mapsto 5 \\ b \mapsto 6 \end{array}, \{y\} \right\rangle.$$

Together they define the constraint problem $C = \langle \{c_1, c_2\}, \{x,y\}\rangle$.

The two main operations on constraints are combination $*$ and projection $\Downarrow$.

Definition 77 (Combination, Projection)**.** *For* $t = (d_x)_{x \in con}$ *and* $con' \subseteq con$ *the projection* $t \downarrow^{con}_{con'}$ *is equal to* $(d_x)_{x \in con'}$.

The combination $c_1 * c_2$ *of two constraints* $c_i = \langle def_i, con_i\rangle, i = 1,2$ *is the constraint* $c = \langle def, con_1 \cup con_2\rangle$*, where*

$$def(t) = def_1(t \downarrow^{con_1 \cup con_2}_{con_1}) \otimes def_2(t \downarrow^{con_1 \cup con_2}_{con_2})$$

and$\otimes$ *is the multiplication of the semiring.*

The projection $c \Downarrow_{con'}$ *of a constraint* $c = \langle def, con\rangle$ *to* $con' \subseteq con$ *is* $\langle def', con'\rangle$ *with* $def'(t') = \bigoplus_{\{t \in D^{con} | t \downarrow^{con}_{con'} = t'\}} def(t)$*, where* $\bigoplus$ *is the addition of the semiring.*

Intuitively, combination $*$ is the product ($\otimes$) of constraint values and projection $\Downarrow_{con'}$ is the sum ($\oplus$) over all assignments to the variables in $con \setminus con'$. Combination, as the name says, combines multiple constraints into one, and projection partially evaluates a constraint thus removing variables from the constraint.

Example 25. *The combination* $c_1 * c_2$ *of* c_1, c_2 *is*

$$\left\langle \begin{array}{l} aa \mapsto 1 \otimes 5 \\ ab \mapsto 2 \otimes 6 \\ ba \mapsto 3 \otimes 5 \\ bb \mapsto 4 \otimes 6 \end{array}, \{x,y\} \right\rangle = \left\langle \begin{array}{l} aa \mapsto 1 + 5 \\ ab \mapsto 2 + 6 \\ ba \mapsto 3 + 5 \\ bb \mapsto 4 + 6 \end{array}, \{x,y\} \right\rangle = \left\langle \begin{array}{l} aa \mapsto 6 \\ ab \mapsto 8 \\ ba \mapsto 8 \\ bb \mapsto 10 \end{array}, \{x,y\} \right\rangle.$$

The projection $c_1 \Downarrow \{x\}$ *of* c_1 *down to* $\{x\}$ *is*

$$\left\langle \begin{array}{l} a \mapsto 1 \oplus 2 \\ b \mapsto 3 \oplus 4 \end{array}, \{x\} \right\rangle = \left\langle \begin{array}{l} a \mapsto \min(1,2) \\ b \mapsto \min(3,4) \end{array}, \{x\} \right\rangle = \left\langle \begin{array}{l} a \mapsto 1 \\ b \mapsto 3 \end{array}, \{x\} \right\rangle.$$

Using $*$ and $\Downarrow$, the consistency-level of an SCSP is defined as follows.

Definition 78 (Consistency-Level)**.** *Given an SCSP problem $P = \langle C, con\rangle$, the* best level of consistency *of P is defined as $blevel(P) = (\Pi_{c\in C}\ c) \Downarrow_\emptyset$. Here, Π in the expression $\Pi_{c\in C}\ c$ is used for application of $*$ to all constraints c in C.*

So the consistency level can be computed by first combining all constraints (using $*$) and then projecting onto the empty set (using $\Downarrow_\emptyset$).

Example 26. *The consistency level of $C = \langle\{c_1, c_2\}, \{x, y\}\rangle$ is $(c_1 * c_2) \Downarrow_\emptyset$, i.e.,*

$$\left\langle \begin{array}{c} aa \mapsto 6 \\ ab \mapsto 8 \\ ba \mapsto 8 \\ bb \mapsto 10 \end{array} , \{x, y\} \right\rangle \Downarrow_\emptyset = \left\langle\ \varepsilon \mapsto \min(6, 8, 8, 10)\ , \emptyset \right\rangle.$$

Here, the last expression evaluates to 6, meaning that the consistency level $blevel(C)$ is 6.

Again, we observe a similar structure regardless of the semiring: we can compute the *blevel* of a constraint problem $P = \langle C, con\rangle$ by guessing an assignment to the variables in *con*, by evaluating the constraints under the given assignment, taking the product, $\otimes$, of their values for that assignment and aggregating the results for each assignment using the sum, $\oplus$, of the semiring. This structure is strikingly similar to the approach we can use to evaluate AMC instances. While the guessed part here is an assignment to multi-valued variables instead of an interpretation, for both frameworks, we can evaluate an instance by

1. *Guessing* some form of assignment;
2. *Evaluating* the instance for the given assignment; and
3. *Aggregating* the values of all guesses.

Using this insight into the structure of the evaluation problems associated with semiring frameworks, we can now approach the characterization of their complexity.

3.4 Semiring Complexity Classes and a Complete Problem

In this section, we develop a toolbox that first and foremost allows completeness results for arbitrary commutative semirings for the first time. For this, we first introduce a prototypical problem called SAT($\mathcal{R}$) using the insights from the previous sections on what the evaluation problems over semiring frameworks typically look like. The reason for introducing a new problem instead of using AMC or SCSPs as a canonical complete problem is that SAT($\mathcal{R}$) in our opinion better shows the power of semiring frameworks.

Rather than only allowing that we compute the value of an assignment as a product of weights, it allows us to specify the value of an assignment as a complex arithmetic expression that depends on a propositional interpretation.

In order to capture this guess-evaluate-aggregate pattern over different semirings, we introduce Semiring Turing Machines (SRTMs), a novel machine model that allows restricted non-deterministic computation of a semiring value in a black box fashion, whose final output is then defined as the sum of the values of all computation paths. Based on SRTMs, we can then define NP($\mathcal{R}$), a generalization of NP over semirings.

Putting things together, we prove SAT($\mathcal{R}$) to be NP($\mathcal{R}$)-complete under Karp reductions, thus allowing us to use it as a canonical problem to reduce from, for NP($\mathcal{R}$)-hardness proofs.

Furthermore, we consider other alternative ways to generalize NP to a quantitative setting. We highlight the differences in computational power caused already by small changes to our model and, by this, justify the definition of NP($\mathcal{R}$) that we chose.

3.4.1 Weighted Quantified Boolean Formulas and SAT($\mathcal{R}$)

The most natural way to define SAT($\mathcal{R}$) is as a special case of weighted Quantified Boolean Formulas (QBFs).

We define weighted QBFs similarly to other Weighted Logics [DG07; MR15] by slightly extending our previous definition of propositional Weighted Logic in Definition 6.

Definition 79 (Syntax)**.** *Let $\mathcal{V}$ be a set of propositional variables and $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ be a commutative semiring. A* weighted *QBF over $\mathcal{R}$ is of the form α given by the grammar*

$$\alpha ::= k \mid v \mid \neg v \mid \alpha + \alpha \mid \alpha * \alpha \mid \Sigma v \alpha \mid \Pi v \alpha$$

where $k \in R$ and $v \in \mathcal{V}$. A variable $v \in \mathcal{V}$ is free *in a weighted QBF α, if there is an occurrence of v in α that is not in the scope of a quantifier Σv or Πv. A weighted* fully quantified Boolean Formula *is a weighted QBF without free variables.*

Definition 80 (Semantics)**.** *Given a weighted QBF α over a commutative semiring $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ and variables from $\mathcal{V}$ as well as a propositional interpretation $\mathcal{I}$ of $\mathcal{V}$, the semantics $[\![\alpha]\!]_\mathcal{R}(\mathcal{I})$ of α over $\mathcal{R}$ w.r.t. $\mathcal{I}$ is defined as follows:*

$$[\![k]\!]_\mathcal{R}(\mathcal{I}) = k$$

$$[\![a]\!]_\mathcal{R}(\mathcal{I}) = \begin{cases} e_\otimes & a \in \mathcal{I} \\ e_\oplus & \textit{otherwise.} \end{cases}$$

$$[\![\neg a]\!]_\mathcal{R}(\mathcal{I}) = \begin{cases} e_\oplus & a \in \mathcal{I} \\ e_\otimes & \textit{otherwise.} \end{cases}$$

$$[\![\alpha_1 + \alpha_2]\!]_\mathcal{R}(\mathcal{I}) = [\![\alpha_1]\!]_\mathcal{R}(\mathcal{I}) \oplus [\![\alpha_2]\!]_\mathcal{R}(\mathcal{I})$$

$$[\![\alpha_1 * \alpha_2]\!]_\mathcal{R}(\mathcal{I}) = [\![\alpha_1]\!]_\mathcal{R}(\mathcal{I}) \otimes [\![\alpha_2]\!]_\mathcal{R}(\mathcal{I})$$

$$[\![\Sigma v\alpha]\!]_{\mathcal{R}}(\mathcal{I}) = [\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}_v) \oplus [\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}_{\neg v})$$
$$[\![\Pi v\alpha]\!]_{\mathcal{R}}(\mathcal{I}) = [\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}_v) \otimes [\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}_{\neg v})$$

where $\mathcal{I}_v = \mathcal{I} \cup \{v\}$ *and* $\mathcal{I}_{\neg v} = \mathcal{I} \setminus \{v\}$.

Weighted QBFs generalize QBFs in negation normal form (NNF), as negation is only allowed in front of variables. Intuitively, allowing negation in front of complex formulas would add the ability to test whether the value of a weighted QBF is zero. We can only test whether an atomic formula is false. Therefore, our variant is or at least seems less expressive. However, it fits better into the context of the problems we consider, where it also is not possible to perform such "zero-tests" for complex expressions.

Here, we further focus on ΣBFs, i.e., the weighted fully quantified BFs that contain only sum quantifiers (i.e. Σv) and we introduce their evaluation problem as

Problem: SAT($\mathcal{R}$)
Input: A ΣBF α over the fixed semiring $\mathcal{R}$
Output: The semantics $[\![\alpha]\!]_{\mathcal{R}}(\emptyset)$

Even though we restrict ourselves to ΣBFs, they at first glance seem to be different from the propositional formulas describing a SAT-instance. Namely, for SAT the quantifier Σ is implicit since all variables that occur in the SAT-instance are existentially quantified. For SAT($\mathcal{R}$), however, this is not the case. Here, we can quantify a variable using Σ in subexpressions of $+$ and $*$, e.g., formulas like $\alpha + \Sigma v\beta$ and $\Sigma v\alpha * \Sigma w\beta$ are allowed. Requiring all quantifiers to occur outside of subexpressions of $+$ and $*$ is not a semantic restriction however:

Lemma 81 (Prefix Normal Form)**.** *For every ΣBF* α *over a semiring* $\mathcal{R}$ *there exists a ΣBF* β *over* $\mathcal{R}$ *such that*

(i) $\beta = \Sigma v_1 \dots \Sigma v_n \gamma$, *where* γ *is quantifier free,*

(ii) β *can be constructed from* α *in polynomial time, and*

(iii) $[\![\alpha]\!]_{\mathcal{R}}(\emptyset) = [\![\beta]\!]_{\mathcal{R}}(\emptyset)$, *i.e.,* α *and* β *evaluate to the same value.*

Proof (Sketch, see Appendix B.1 for the full proof). This statement can be proved by induction on the number of sum quantifiers Σ that occur in the subexpression of some other connective $+$ or $*$. The base case is clear, for the induction step, we show that $(\Sigma v\alpha_1) * \alpha_2$ is equivalent to $\Sigma v(\alpha_1 * \alpha_2)$ by assuming w.l.o.g. that α_2 does not contain v and that $(\Sigma v\alpha_1) + \alpha_2$ is equivalent to $\Sigma v(\alpha_1 + \alpha_2 * v)$ under the same assumption. □

This way the connection between SAT and SAT($\mathcal{R}$) becomes clearer. In fact, as with AMC, many versions of SAT can be seen as special cases of SAT($\mathcal{R}$) for different semirings $\mathcal{R}$.

Example 27. *Over $\mathbb{B}$, the Boolean semiring, SAT($\mathbb{B}$) is equivalent to SAT: On the one hand, given a SAT($\mathcal{R}$)-instance $\beta = \Sigma v_1 \dots \Sigma v_n \gamma$, where γ is quantifier free, we see that $[\![\beta]\!]_{\mathbb{B}}(\emptyset) = 1$ iff the propositional formula ϕ obtained from γ by replacing every $0, 1, +, *$ with $\bot, \top, \vee, \wedge$, respectively, is a yes instance of SAT, i.e., satisfiable.*

On the other hand, given a SAT-instance ϕ, we see that a propositional formula ϕ in NNF is satisfiable if the SAT($\mathbb{B}$)-instance $\beta = \Sigma v_1 \dots \Sigma v_n \gamma$ fulfills $[\![\beta]\!]_{\mathbb{B}}(\emptyset) = 1$. Here, γ is obtained from ϕ by replacing every $\bot, \top, \vee, \wedge$ with $0, 1, +, *$, respectively, and $\{v_1, \dots, v_n\}$ is the set of variables occurring in ϕ.

Example 28. *Another, more interesting example is LexMaxSAT, the problem of obtaining the lexicographically maximum satisfying assignment for a propositional formula ϕ. W.l.o.g. we assume that ϕ is in NNF. We can reformulate LexMaxSAT as a SAT($\mathcal{R}_{\max,+}$)-instance β over the max-tropical semiring $\mathcal{R}_{\max,+}$, where the corresponding instance β is given by*

$$\Sigma v_1 \dots \Sigma v_n \gamma * (v_1 * 2^{n-1} + \neg v_1) * \dots * (v_n * 2^0 + \neg v_n).$$

As above, γ is obtained from ϕ by replacing every $\bot, \top, \vee, \wedge$ with $-\infty, 0, \max, +$, respectively. Then $[\![\beta]\!]_{\mathcal{R}_{\max}}(\emptyset) = e_{\oplus} = -\infty$ iff ϕ is unsatisfiable and $[\![\beta]\!]_{\mathcal{R}_{\max}}(\emptyset) = m = \sum_{i=1}^{n} b_i 2^{n-i}$ iff the lexicographically maximum satisfying assignment for ϕ sets v_i to true whenever $b_i = 1$.

These examples show that we can use SAT($\mathcal{R}$) over different semirings $\mathcal{R}$ to express different versions of SAT. Moreover, we observe that problems described by a ΣBF of the form

$$\beta = \Sigma v_1 \dots \Sigma v_n \gamma, \text{ where } \gamma \text{ is quantifier free,}$$

are again structurally similar to the other problems following the semiring paradigm. That is, the value $[\![\beta]\!]_{\mathcal{R}}(\emptyset)$ is obtainable by (i) *guessing* an interpretation $\mathcal{I}$ of the variables $v_1, \dots, v_n$, (ii) *evaluating* γ using the interpretation, and (iii) *aggregating* the values $[\![\gamma]\!]_{\mathcal{R}}(\mathcal{I})$ using the sum, $\oplus$, of the semiring.

This guess-evaluate-aggregate pattern is also similar to definition of complexity classes such as NP, #P and OptP in terms of non-deterministic Turing machines. Based on this intuition we will next provide NP($\mathcal{R}$) a generalized version of NP, which also depends on a semiring parameter, to abstractly characterize the complexity of SAT($\mathcal{R}$).

3.4.2 Semiring Turing Machines and NP($\mathcal{R}$)

We generalize NTMs to Semiring Turing Machines (SRTMs) to characterize the complexity of SAT($\mathcal{R}$). At this point, we do not want the time and space needed to perform operations on semiring values that are encoded in a finite alphabet to influence our definition. Thus, we need SRTMs to be capable of

- performing semiring operations atomically in a black box manner, irrespective of encodings of values;

- summing up values generated by nondeterministic computations; and
- using semiring values in the input in calculations.

On the other hand, too much power should be avoided; to this end, we relegate semiring computations to weighted transitions.

Definition 82 (SRTM)**.** *A* Semiring Turing Machine *is a 7-tuple* $M = (\mathcal{R}, R', Q, \Sigma, \iota, \sqcup, \delta)$*, where*

- $\mathcal{R}$ *is a commutative semiring,*
- $R' \subseteq R$ *is a finite set of semiring values, (intuitively, this is a set of fixed values that are "known" to* M*)*
- Q *is a finite set of states,*
- Σ *is a finite set of symbols (the tape alphabet),*
- $\iota \in Q$ *is the initial state,*
- $\sqcup \in \Sigma$ *is the blank symbol,*
- $\delta \subseteq (Q \times (\Sigma \cup R)) \times (Q \times (\Sigma \cup R)) \times \{-1, 1\} \times R$ *is a weighted transition relation, where the last entry of the tuple is the weight. For each* $((q_1, \sigma_1), (q_2, \sigma_2), d, r) \in \delta$ *the following holds:*

  1. M cannot write or overwrite semiring values*:*
     if $\sigma_1 \in R$ *or* $\sigma_2 \in R$*, then* $\sigma_1 = \sigma_2$*,*
  2. M can only make a transition with weight r when r is from R' or under the head:
     $r \in R'$ *or* $r = \sigma_1 \in R$
  3. M cannot discriminate semiring values:
     if $\sigma_1 \in R$*, then*
     a) for all $\sigma_1' \in R$ *we have* $((q_1, \sigma_1'), (q_2, \sigma_1'), d, \sigma_1') \in \delta$ *or*
     b) for all $\sigma_1' \in R$ *we have* $((q_1, \sigma_1'), (q_2, \sigma_1'), d, r) \in \delta$*.*

As usual, -1 and 1 move the head to the left and right. Intuitively, the combination of the second and third condition ensures that when an SRTM reads a semiring value r in a given state then it always, independently of the value at hand, makes a transition with weight r or with a constant value from the finite set R'. Note that this "or" is not exclusive. Having access to some constants is necessary, since every transition is assumed to have a weight. Additionally, these constants allow us to use semiring values that do not occur in the input. This is important since otherwise we cannot always multiply partial solutions by constant factors or even return a constant value.

Importantly, this allows us to always represent the transition function finitely by grouping the transitions at state q_1 when a semiring value is read to

$$((q_1, X), (q_2, X), d, X) \text{ and } ((q_1, X), (q_2, X), d, r),$$

where X is a placeholder representing any (although the same in all occurrences in the expression) semiring value and $r \in R'$.

We remark that in terms of control flow, i.e., "if"-statements, loops, recursive definitions, etc., SRTMs feature the same possibilities as non-deterministic Turing machines, since the difference between the two models of computation regarding the transition function δ is that for SRTMs it is weighted and has some restrictions on the weights. Therefore, we can use the typical control flow instructions also with SRTMs.

The output of a computation is as follows.

Definition 83 (SRTM function)**.** *A* configuration *is a triple $c = (q, x, n)$, where $q \in Q$ is a state, $x \in (\Sigma \cup R)^*$ is the string on the tape, and $n \in \mathbb{N}$ is the head position. The* value $v(c)$ *of an SRTM M on configuration $c = (q, x, n)$, is recursively defined by $v(c) = \bigoplus_{c \xrightarrow{r} c'} r \otimes v(c')$, where $c \xrightarrow{r} c'$ denotes that M can transit from c to the next configuration c' with weight r; the empty sum has value $e_\otimes$. The* output *of M on input x is $v(\iota, x, 0)$, the value of the initial configuration.*

In other words, this means that the output of an SRTM M is calculated by aggregating the value v_π of each non-deterministic computation path π using the addition $\oplus$ of the semiring $\mathcal{R}$. Here, the value v_π of a non-deterministic computation path π along configurations $c_1 \xrightarrow{r_1} \dots \xrightarrow{r_{n(\pi)-1}} c_{n(\pi)}$ is given by $r_1 \otimes \dots \otimes r_{n(\pi)-1}$, i.e., by multiplying the weights of the single transition steps.

Example 29 (SRTM Computation)**.** *Consider Algorithm 1, which sketches in pseudocode the working of an SRTM that solves SAT($\mathcal{R}$). The computation tree*[2] *of* $\textsc{Eval}_{\mathbb{N}}(\alpha, \emptyset)$ *with $\alpha = \Sigma v v * 3$ is given in Figure 3.1. Note that here we only consider configurations abstractly in the sense that if we were to translate the pseudocode into an SRTM there would be actual configurations to replace them.*

The initial configuration c_0 is the root node of the tree. Since the SRTM is over the natural number semiring $\mathbb{N}$, the leaves c_5, c_6, i.e., the configurations without a successor, have the value $e_\otimes = 1$.

The values of their predecessors c_3 and c_4 are both $3 \cdot 1 = 3$. It is calculated as the value of their successor (1) and the weight of the transition, which is 3 in both bases since the SRTM reads the subformula $\alpha = 3$ in either case. At their predecessors c_1 and c_2, we observe a difference in values, i.e., $v(c_1) = 0$ and $v(v_{0.1}) = 3$, since the SRTM reads the subformula $\alpha = v$ here and thus transitions either with weight $e_\otimes$ or $e_\oplus$ depending on the guess of the value of v in the respective branch. c_1 and c_2 have as a common predecessor

[2] Naturally, we left out transitions with weight 1 for the checks of the if and switch statements.

Algorithm 1 An SRTM algorithm for SAT($\mathcal{R}$)

Input A ΣBF α over semiring $\mathcal{R}$ and an interpretation $\mathcal{I}$.
Output $[\![\alpha]\!]_{\mathcal{R}}(\mathcal{I})$.

```
1:  function EVAL_R(α, I)
2:      switch α do
3:          case α = k:
4:              transition with k
5:          case α ∈ {v, ¬v}:
6:              if (α = v and v ∈ I) or (α = ¬v and v ∉ I) then:
7:                  transition with e_⊗
8:              else:
9:                  transition with e_⊕
10:         case α = α_1 + α_2:
11:             Guess i ∈ {1, 2}
12:             Execute EVAL_R(α_i, I)
13:         case α = α_1 * α_2:
14:             Execute EVAL_R(α_1, I)
15:             Execute EVAL_R(α_2, I)
16:         case α = Σvα':
17:             Guess I' ∈ {I ∪ {v}, I \ {v}}
18:             Execute EVAL_R(α', I')
```

the initial configuration c_0. Here, a formula of the form $\alpha = \Sigma v\alpha$ is read and thus both transitions from c_0 to c_1 and to c_2, have weight 1. *They correspond to guessing v to be false and true, respectively. As such, the output value is given by* 3.

As we can see, this definition of SRTMs also follows the aforementioned guess-evaluate-aggregate pattern. With this in place, we define an analog of NP.

Definition 84 (NP($\mathcal{R}$))**.** *NP($\mathcal{R}$) is the class of all functions computable in polynomial time by an SRTM over $\mathcal{R}$.*

Here, we choose the name NP($\mathcal{R}$) to highlight the tight connection to NP, which stems from the fact that (as we will see later) NP can be seen as NP($\mathbb{B}$). Indeed, the underlying idea of SRTMs and NP($\mathcal{R}$) is to proceed in the same manner as when generalizing a Boolean logic to a weighted logic. That is, we add weights and replace disjunction by addition and conjunction by multiplication over the semiring. Since over the Boolean semiring $\mathbb{B}$ the addition is disjunction and the multiplication is conjunction, solving a problem in NP($\mathbb{B}$) is to check whether there exists a non-deterministic computation path with non-zero weight, i.e., which accepts. The same holds for problems in NP.

Note that NP($\mathcal{R}$) as it follows from our definition of SRTMs is only *some* analog of NP over semirings, and we do not claim that it is the only reasonable one. However, first and

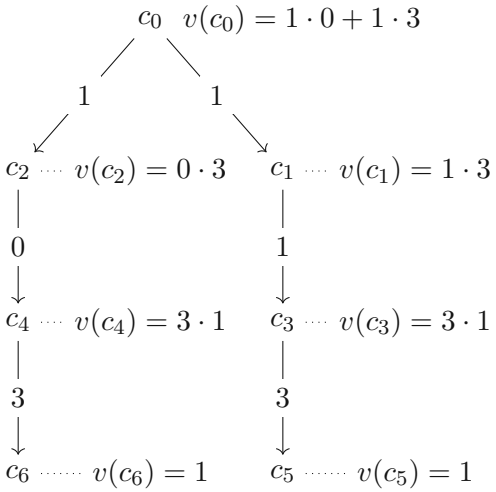


Figure 3.1: A computation tree over $\mathbb{N}$. Each transition $c \xrightarrow{r} c'$ is annotated with its weight r and each configuration c is annotated with its value $v(c)$.

foremost, our goal with this definition is to obtain a complexity class that characterizes the computational complexity of $\mathrm{SAT}(\mathcal{R})$ and other related problems. This succeeds, as indeed $\mathrm{SAT}(\mathcal{R})$ is $\mathrm{NP}(\mathcal{R})$-complete. However, in order to formally state the result we first need to establish how reductions work in this context, given that we allow semiring values on the tape explicitly.

Definition 85 (Surrogate Alphabet)**.** *Let $\mathcal{R}$ be a semiring and let Σ be some alphabet. Then the* surrogate alphabet $s_{S,V,E}(\Sigma)$ *is set of words over Σ extended with surrogates (for semiring variables) defined by*

$$s_{S,V,E}(\Sigma) = (\Sigma \cup \{SV^nE \mid n \in \mathbb{N}\})^*,$$

where $S, V, E \notin \Sigma$ are distinguished letters such that in SV^nE the letter S denotes the start, the string V^n denotes the index n, and E denotes the end of a surrogate (for a semiring value). For a substitution $\sigma : \mathbb{N} \to R$ and a word $x \in s_{S,V,E}(\Sigma)$, we denote by $\sigma(x)$ the word in $(\Sigma \cup R)^*$ obtained by replacing SV^nE with $\sigma(n)$.

The idea here is that we refer to surrogates $SVE, \ldots, SV^nE$ for variable semiring values instead of actual semiring values $r_1, \ldots, r_n$. Thus, we know of the presence of semiring values but not which semiring value is represented exactly by the surrogate.

Using the surrogate alphabet, we adapt Karp reductions to the context of SRTMs.

Definition 86 (Karp Surrogate-Reduction)**.** *Let f_1 and f_2 be functions $f_i : (\Sigma \cup R)^* \to R, i = 1, 2$. Then a Karp* surrogate $s_{S,V,E}$ reduction *from f_1 to f_2 is a polynomial time computable function $T : s_{S,V,E}(\Sigma) \to s_{S,V,E}(\Sigma)$ with a finite set $R' = \{r_1, \ldots, r_k\} \subseteq R$, $k \geq 0$, of semiring values such that for all $x \in s_{S,V,E}(\Sigma)$ and $\sigma : \mathbb{N} \to R$ such that $\sigma(i{-}1) = r_i$, $1 \leq i \leq k$, it holds that $f_1(\sigma(x)) = f_2(\sigma(T(x)))$.*

Since we only use Karp surrogate $s_{S,V,E}$ reductions with surrogate alphabet S, V, E we omit $s_{S,V,E}$ and write "Karp s-reduction".

Intuitively, the idea behind surrogate-reductions is that as with SRTMs, we cannot modify semiring values, make decisions based on them etc. Therefore, also surrogate reductions must not be able to modify semiring values, produce different outputs based on them etc., as this would mean that reductions would have more power than SRTMs, which we need to avoid if $\mathrm{NP}(\mathcal{R})$ should be closed under reductions. The set R' serves to distinguish semiring values that should be constant. Instead we can only modify the instance specification that tells us how we should combine (using sum and product) the semiring values to obtain a solution from one that is expected by f_1 to one that is expected by f_2.

Nevertheless, a reduction T at least needs to be able to transfer the semiring values that are used in the original instance x to the instance $T(x)$. For this reason, we give T access to surrogates SV^nE for semiring value occurrences such that T can intuitively "copy" semiring values to another position but cannot do more.

While this is a strong restriction, we will show in Theorem 88 that this suffices to reduce any problem in $\mathrm{NP}(\mathcal{R})$ to $\mathrm{SAT}(\mathcal{R})$ for any commutative semiring $\mathcal{R}$.

Surrogate-reductions are appropriate for our setting:

Lemma 87. *Let $\mathcal{R}$ be a semiring and $f_i : (\Sigma \cup R)^* \to R, i = 1, 2$. If $f_2 \in NP(\mathcal{R})$ and f_1 is Karp s-reducible to f_2 then $f_1 \in NP(\mathcal{R})$.*

Proof. Let T be the reduction function and $R' = \{r_1, \ldots, r_k\}$ the associated set of semiring constants. Let $M = (\mathcal{R}, R'_M, Q, \Sigma, \iota, \sqcup, \delta)$ be an SRTM that solves f_2. We construct an SRTM $M' = (\mathcal{R}, R'_M \cup R', Q', \Sigma', \iota', \sqcup', \delta')$ that solves f_1. Note that we give M' access to both R'_M, the set of semiring constants from M, and R', the set of semiring constants, fixed in the reduction.

While SRTMs cannot distinguish different semiring values, they can check whether there is a semiring value on the tape in a given position. Thus, given some input $x \in (\Sigma \cup R)^*$ we can obtain in polynomial time a string $x' \in s_{S,V,E}(\Sigma)$ from x by using $SV^{n+k}E$ for the n^{th} semiring value occurrence in x (from left to right). E.g. the string $x = x_1x_2x_3r_1x_4x_5r_2 \in (\Sigma \cup R)^*$, where $x_1, \ldots, x_4 \in \Sigma$ and $r_1, r_2 \in R$, leads to the string $x' = x_1x_2x_3SVEx_4x_5SVVE \in s_{S,V,E}(\Sigma)$ when $k = 0$. Then, we can apply T to x' in polynomial time. Finally, we can execute the algorithm for f_2 on $T(x')$. Here, we only need polynomial extra time to look up the $(n-k)^{\text{th}}$ semiring value in the original input when we need to make a transition with a semiring value under the head and read SV^nE for $n > k$. For $n \leq k$ we cannot transition with a weight under the head, since r_n does not necessarily occur in x. However, since we added R' to the set of constants that M' has access to, we can simply specify a transition with weight r_n directly. □

With this in mind, we can finally state the main result of the section.

Theorem 88. *SAT($\mathcal{R}$) is NP($\mathcal{R}$)-complete w.r.t. Karp s-reductions for every commutative semiring $\mathcal{R}$.*

Proof (Sketch, see Appendix B.2.1 for the full proof). We prove membership constructively, using the SRTM Algorithm 1. We need to prove that $\textsc{Eval}_{\mathcal{R}}(\alpha, \emptyset) = [\![\alpha]\!]_{\mathcal{R}}(\emptyset)$, for any ΣBF α. When considering the formula α we could exhaustively apply the distributive law for every addition $(+, \Sigma a)$ that occurs inside a multiplication $(*)$. Then one obtains a sum of products of values from the semiring and literals. In general this would, however, lead to a formula that is exponentially bigger than α. Instead, the idea of $\textsc{Eval}_{\mathcal{R}}$ is to non-deterministically generate every term s that we *would* have after having exhaustively applied the distributive law. For each non-deterministically generated s consisting of factors $f_1, \ldots, f_m$, $\textsc{Eval}_{\mathcal{R}}$ has a computation path that includes a transition with weight f_i for each $i = 1, \ldots, m$. This happens in such a way that each sequence of choices leads to a different s and such that every s is covered by a sequence of choices. Thus, the sum, using $\oplus$, of the value of all execution paths for a call to $\textsc{Eval}_{\mathcal{R}}(\alpha, \emptyset)$ is equal to $[\![\alpha]\!]_{\mathcal{R}}(\mathcal{I})$.

To prove the NP($\mathcal{R}$)-hardness, we can generalize the well-known Cook-Levin Theorem, cf. [GJ79]. Here, we need to rewrite disjunctions $a \vee b$ into formulas $a + \neg a * b$ s.t. these constraints can only be equal to either $e_\oplus$ or $e_\otimes$ but not $e_\otimes \oplus e_\otimes$. Apart from that we only need to add the weights of transitions to the formula. □

Nevertheless, before we prove a wide range of semiring formalism to be NP($\mathcal{R}$)-complete, we discuss in the following possible alternative choices in the definition of SRTMs and their effect on the computational power of SRTMs.

Alternative SRTM Definitions

There are multiple possible small changes in the definition of SRTMs that would lead to an alternative model of non-deterministic computations over semirings and therefore also to an alternative complexity class NP($\mathcal{R}$). We discuss some of these possible changes and their implications on the power of SRTMs, to explain our rationale for choosing this definition of SRTMs.

Allowing Semiring Operations on the Tape One alternative would be to remove the weights from the transition functions and instead to let the machine multiply and add values on a tape explicitly. We see two main options that would enable this change. The first is to depart from the idea to use semiring values on the tape and to encode them in some finite alphabet instead. This would have the positive effect of getting a definition that is very close to the definition of typical non-deterministic TMs. However, this comes with two significant drawbacks.

First, it implies that we need to choose an encoding of the semiring values. As we will see later on, in Examples 34 and 35, the choice of the encoding can strongly influence the complexity of semiring operations. We want to avoid this, since we want to use SRTMs

to characterize the complexity associated with a semiring, rather than the complexity associated with the encoding of its values.

Second, it would allow for the computation of functions that cannot be Karp-reduced to SAT($\mathcal{R}$).

Example 30 (Separating Function I)**.** *The function* $f : \{0,1\}^* \to \mathbb{N}[(x_i)_\infty]$, $f(w) = x_1 + x_2 + \cdots + x_{2^{|w|}}$ *cannot be Karp reduced to NP(*$\mathbb{N}[(x_i)_\infty]$*) using an encoding* $e : \mathbb{N}[(x_i)_\infty] \to \Sigma^*$ *in some appropriate alphabet* $\Sigma = \{0,1\} \cup \{\dots\}$*, where coefficients, exponents and indices are encoded in binary, and a polynomial is encoded as a list of the encodings of the monomials. Assume that on the contrary,* α_w *is the SAT(*$\mathbb{N}[(x_i)_\infty]$*)-instance corresponding to input* w *to* f*. Then* α_w *needs to contain each of the monomials* $x_1, \dots, x_{2^{|w|}}$ *and is therefore necessarily of size exponential in* $|w|$*. Since Karp reductions must be computable in polynomial time, this is impossible.*

On the other hand, when we explicitly write semiring values in encoded form on the alphabet of a TM, we can guess $1 \leq i \leq 2^{|w|}$ *and return* x_i*. Then the sum of all computation paths is* $f(w)$*.*

Alternatively, we could allow for the computation of sums/products of pairs of semiring values, which can be written to the tape (e.g. using multiple tapes or heads). Without additional restrictions, this would also give SRTMs more power than what can be expressed using SAT($\mathcal{R}$).

Example 31 (Separating Function II)**.** *Consider the machine over the semiring* $\mathbb{N}$ *that takes as input a pair* (k, x)*, where* $k \in \mathbb{N}$ *and* $x \in \{0,1\}^*$ *has length* $|x| = n$*, and computes a semiring value as output as follows: first, it sets* $x_1 = k$ *and then iteratively sets* $x_{i+1} = x_i * x_i$ *for* $i = 2, \dots, n$*. Then the output of the machine is* $x_n = k^{2^n}$*. This computation is possible if we allow computations on the tape as described above. On the other hand, every Karp s-reduction from* (k, x) *to a SAT(*$\mathbb{N}$*)-instance* α *satisfies that* $[\![\alpha]\!]_{\mathbb{N}}(\emptyset)$*, the result of evaluating* α*, is in* $\mathcal{O}(k^{\mathcal{O}(n^c)} \cdot 2^{\mathcal{O}(n^c)})$*, where* c *is a constant. This is because we can only use the semiring value* k *in* α *when using s-reductions and since we know that any SAT(*$\mathbb{N}$*)-instance* α *has a value in* $\mathcal{O}(\max_{r \in \alpha} r^{|\alpha|} \cdot 2^{|\alpha|})$*.*

Note that interestingly, in a circuit version of SAT($\mathbb{N}$) we *are* able to express the value k^{2^n} for $k \in \mathbb{N}$ using an instance of size polynomial in n as they also allow reuse of partial results multiple times. This also implies that even though CircuitSAT and SAT are Karp-reducible to one another, the same is in general not the case for SAT($\mathcal{R}$) and its circuit version.

Nevertheless, by restricting the semiring computations on the tape to be *non-recursive*, i.e., by only allowing the use of intermediate results only once, this problem is avoided. Note that as the above example shows allowing the use of intermediate results *twice* already leads to a problem. But with this restriction in place the two definitions would result in two machine models with the same power that can simulate each other. Since,

however, the non-recursiveness restriction is a semantic rather than a syntactic one (and thus harder to verify and enforce) we prefer our original definition.

Allowing Decisions Based on Semiring Values Recall that Condition 3. on the weighted transition function δ requires that for each $((q_1, \sigma_1), (q_2, \sigma_2), e, r) \in \delta$,

$$\sigma_1 \in R \Rightarrow \ [\forall \sigma_1' \in R : ((q_1, \sigma_1'), (q_2, \sigma_1'), e, \sigma_1') \in \delta] \ \vee \ [\forall \sigma_1' \in R : ((q_1, \sigma_1'), (q_2, \sigma_1'), e, r) \in \delta].$$

Therefore, we cannot distinguish different semiring values that are on the tape *during* the computation. That is, we cannot draw conclusions about the semiring value on the tape based on the state q_2 we transition to or based on the symbols on the tape that are non-semiring values. Furthermore, we cannot distinguish semiring values on the tape based on the weight of the transitions they are involved in, if the weight is not the semiring value itself but a constant from R'. SRTMs without this property may not necessarily be finitely specified. Apart from that, having the ability to take different transitions based on the semiring value would allow an SRTM over semiring values from $\{0, 1\}^*$ to distinguish strings that encode halting Turing machines from strings that do not, in constant time. This is obviously not desirable.

Allowing Semiring Values to be Written Also the first restriction on the weighted transition relation, which constitutes that we cannot write new semiring values to the tape or change existing ones, may seem questionable at first glance. Allowing to write or change semiring values in an unrestricted manner would allow us to distinguish semiring values. We already described previously why this would be undesirable. However, it is possible to allow for writing and changing of semiring values on the tape under restrictions such as only allowing to copy values or write values from a finite set. This would neither lead to additional power nor would it make the specification of SRTM machines simpler. Therefore, we chose this simpler version of the definition of SRTMs and only note that this restricted form of writing semiring values can be simulated.

3.5 Completeness Results for Semiring Frameworks

SAT($\mathcal{R}$) and NP($\mathcal{R}$) generalize SAT and NP to the semiring setting. As we discussed above their definitions equip SAT($\mathcal{R}$) and NP($\mathcal{R}$) with appropriate power. This is underlined also by the fact that SAT($\mathcal{R}$) is NP($\mathcal{R}$)-complete as one might expect. Therefore, we now have all the necessary prerequisites for a complexity analysis of frameworks that are defined dependent on a semiring parameter.

In the following, we apply the results from above to different semiring formalisms in AI.

3.5.1 Sum-Of-Products Problems

Another prototypical problem that is similar to SAT($\mathcal{R}$) is the Sum-of-Products Problem SumProd($\mathcal{R}$) [BDP09]. It is defined as follows

Definition 89 (SumProd($\mathcal{R}$))**.** *Given a finite domain $\mathcal{D}$ and functions $f_i : \mathcal{D}^{j_i} \to R, i = 1, \ldots, n$, given explicitly as key-value pairs of variable assignments and semiring values, compute*

$$\bigoplus_{X_1,\ldots,X_m \in \mathcal{D}} \bigotimes_{i=1}^{n} f_i(\vec{Y_i}), \tag{3.1}$$

where $\vec{Y_i}$ is a vector with entries from a set $\{X_1, \ldots, X_m\}$ of variables.

To solve a problem instance, we need to compute the sum of the products of the functions f_i for all assignments of the variables X_i. Here, the "sum" and the "product" are the addition $\oplus$ and multiplication $\otimes$ of the semiring $\mathcal{R}$, respectively.

There are two main differences between SAT($\mathcal{R}$) and SumProd($\mathcal{R}$). The first is that SumProd($\mathcal{R}$) does not admit sums that occur under products, whereas for SAT($\mathcal{R}$) sums and products can alternate arbitrarily. Second, SumProd($\mathcal{R}$) is not propositional but more related to first-order logic, since we are not summing up values using a quantifier $\Sigma a \alpha$ by evaluating α under both truth values of a. Instead, we are summing up values over all assignments to variables over a finite domain.

Nevertheless, we obtain:

Theorem 90. *SAT($\mathcal{R}$) is Karp s-reducible to SumProd($\mathcal{R}$) and vice versa for every commutative semiring $\mathcal{R}$. Thus, SumProd($\mathcal{R}$) is NP($\mathcal{R}$)-complete w.r.t. Karp s-reductions.*

In the Boolean case, i.e. $\mathcal{R} = \mathbb{B}$, SumProd($\mathcal{R}$) essentially corresponds to Constraint Satisfaction Problems (CSP). Here, the reduction from SAT($\mathcal{R}$) to SumProd($\mathcal{R}$) borrows from the fact that 3SAT is NP-complete. However, for semirings in general establishing this result is more difficult, since the Tseitin-transformation [Tse83], which is used for the proof that 3SAT is NP-complete, does not work anymore in the same way. The idea of the Tseitin-transformation is to introduce a new variable a_β for a complex subformula β such that a_β takes the value of β. In the Boolean case, we recall that this is established by means of the formula $a_\beta \leftrightarrow \beta$. In our case, the value of a subformula β may not only be $\top$ or $\bot$ as in the Boolean case but can take one of exponentially many different values in the size of β. So, while we could introduce variables $a_{\beta,r}$ that are true iff β evaluates to r, this would lead to an exponential explosion and is, thus, not helpful for proving the existence of a Karp s-reduction.

Instead, we need to exploit the distributive law. While naive application of it also leads to a formula of exponential size, we can avoid this by exploiting the distributivity implicitly during evaluation as in Algorithm 1. Intuitively, the idea here is to introduce new variables a_{β_1}, a_{β_2} for each non-deterministic choice introduced by a subformula $\alpha = \beta_1 + \beta_2$, which tell us whether we include β_1 or β_2. A detailed proof is given in Appendix B.3.3.

Note that it is not strictly necessary that the functions f_i are given explicitly as key-values pairs. In fact, it suffices if f_i is computable in polynomial time by an SRTM, since we

can simulate this computation during SAT($\mathcal{R}$)-evaluation. On the other hand, it is not clear whether it is sufficient that f_i produces a binary representation of semiring values since there is no straight-forward way of simulating this kind of manipulation during SAT($\mathcal{R}$)-evaluation (or using SRTMs for that matter).

3.5.2 Algebraic Constraints

Weighted first-order logic was introduced by Mandrali and Rahonis [MR15] for expressivity characterizations. We furthermore used them in Section 2.3 to define algebraic constraints for a quantitative extension of ASP. For simplicity's sake we reintroduce them here without sorts and non-monotonicity through Here-and-There semantics.

It is defined over a *signature* $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X} \rangle$ with predicates $p \in \mathcal{P}$ that have a fixed arity $\text{arity}(p) \in \mathbb{N}$ over a domain $\mathcal{D}$ and variables in $\mathcal{X}$.

Definition 91 (Syntax)**.** *Let $\sigma = \langle \mathcal{D}, \mathcal{P}, \mathcal{X} \rangle$ be a signature and $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ be a commutative semiring. The* weighted σ-formulas *over $\mathcal{R}$ are of the form α given by the grammar*

$$\alpha ::= k \mid p(\vec{x}) \mid \neg p(\vec{x}) \mid \alpha + \alpha \mid \alpha * \alpha \mid \Sigma x \alpha \mid \Pi x \alpha.$$

Here $k \in R$, $p \in \mathcal{P}$, $\vec{x} \in (\mathcal{D} \cup \mathcal{X})^{arity(p)}$ and $x \in \mathcal{X}$. A weighted σ-sentence *is a weighted σ-formula that does not contain free variables.*

Note that we again only allow for negation in front of atoms $p(\vec{x})$.

Definition 92 (Semantics)**.** *A* σ-interpretation *is a subset $\mathcal{I}$ of $\{p(\vec{x}) \mid p \in \mathcal{P}$ and $\vec{x} \in \mathcal{D}^{arity(p)}\}$. Given a weighted σ-sentence β and a σ-interpretation $\mathcal{I}$, the semantics $[\![\beta]\!]_\mathcal{R}(\mathcal{I})$ of β over $\mathcal{R}$ w.r.t. $\mathcal{I}$ is defined as*

$$[\![\Sigma x \beta]\!]_\mathcal{R}(\mathcal{I}) = \bigoplus_{d \in \mathcal{D}} [\![\beta\{x \mapsto d\}]\!]_\mathcal{R}(\mathcal{I}),$$
$$[\![\Pi x \beta]\!]_\mathcal{R}(\mathcal{I}) = \bigotimes_{d \in \mathcal{D}} [\![\beta\{x \mapsto d\}]\!]_\mathcal{R}(\mathcal{I}).$$

The rest of the cases are as in Definition 80, where we identify $p(\vec{x})$ with a propositional variable $v_{p(\vec{x})}$.

We consider the evaluation of ΣFO σ-formulas, which only use sum quantifiers (i.e. Σx):

Problem:	ΣFO-Eval($\mathcal{R}$)
Input:	A weighted ΣFO σ-sentence α over the fixed commutative semiring $\mathcal{R}$ and a σ-interpretation $\mathcal{I}$
Output:	The semantics $[\![\alpha]\!]_\mathcal{R}(\mathcal{I})$

Quantitative first-order evaluation problems are present for example in probabilistic inference from Bayesian networks [VMD14]; furthermore, when we want to evaluate an algebraic constraint, over a given interpretation, we are faced with the same problem.

The ΣFO-Eval($\mathcal{R}$) problem is very similar to SAT($\mathcal{R}$). Indeed, under the assumption that $\mathcal{I}$ is given explicitly as key-value pairs of predicates with variable assignments and truth values, we obtain:

Theorem 93. *Problem ΣFO-Eval($\mathcal{R}$) is Karp s-reducible to SAT($\mathcal{R}$), and vice versa, and thus NP($\mathcal{R}$)-complete w.r.t. Karp s-reductions for every commutative semiring $\mathcal{R}$.*

Proof (Sketch, see Appendix B.2.2 for the full proof). $\Rightarrow$: Let α be a ΣBF over $\mathcal{R}$. We choose $\sigma = \langle\{\bot, \top\}, \{t(.)\}, \{x_{v_1}, \ldots, x_{v_n}\}\rangle$ and $\mathcal{I} = \{t(\top)\}$ and replace every propositional variable v in α by $t(x_v)$.

$\Leftarrow$: We replace $p(\vec{x})$ by $\Sigma_{p(\vec{d})\in\mathcal{I}}\Pi_{x_i\in\vec{x}} v_{x_i,d_i}$, where $\vec{x} = x_1, \ldots, x_{\text{arity}(p)}$, $\vec{d} = d_1, \ldots, d_{\text{arity}(p)}$, and $v_{x,d}$ means that variable x has value d. We add constraints such that when both v_{x_i,d_i} and v_{x_i,d'_i} are true, then $d_i = d'_i$ and add a quantifier $\Sigma v_{x_i,d_i}$ for each pair (x_i, d_i). □

3.5.3 Semiring-based Constraint Satisfaction Problems

Recall the definition of SCSPs from Section 3.3. We see that the computation of $blevel(P)$ is a sum of the products of the values of the constraints in P over all possible variable assignments. We obtain:

Theorem 94. *Computing blevel(.) over $\mathcal{R}$ is Karp s-reducible to SumProd($\mathcal{R}$) and vice versa, and thus NP($\mathcal{R}$)-complete w.r.t. Karp s-reductions for every commutative semiring $\mathcal{R}$.*

Proof (Sketch, see Appendix B.2.2 for the full proof). As the variables of the constraint problem we use the ones of SumProd($\mathcal{R}$) and for each constraint $\langle def_i, con_i\rangle$ we use the function def_i as a function f_i in SumProd($\mathcal{R}$); $blevel(.)$ is the solution of SumProd($\mathcal{R}$). □

3.5.4 Algebraic Model Counting

Recall the definition of AMC from Section 3.3. Since AMC also follows the guess-evaluate-aggregate pattern, we can show:

Theorem 95. *AMC over $\mathcal{R}$ is Karp s-reducible to SAT($\mathcal{R}$) and vice versa, and thus NP($\mathcal{R}$)-complete w.r.t. Karp s-reductions for every commutative semiring $\mathcal{R}$.*

The main differences between AMC and SAT($\mathcal{R}$) are that, on the one hand, in AMC the logical part and the part that weights an interpretation are explicitly separated, whereas for SAT($\mathcal{R}$) they are intertwined. Second, while AMC only allows for weight functions that can be expressed as a product of weights, SAT($\mathcal{R}$) allows for complex arithmetic expressions over the semiring.

Proof (Sketch, see Appendix B.2.2 for the full proof). $\Rightarrow$: Given an AMC instance as a propositional theory T and a labeling function $\alpha : L \to R$ over the semiring $\mathcal{R}$, we can translate T into a ΣBF β and weight it with α using the product of the weighted formulas $(v_i * \alpha(v_i) + \neg v_i * \alpha(\neg v_i))$ for $v_i \in \mathcal{V}$.

$\Leftarrow$: As we noticed before, if we were to apply the distributive law exhaustively on a SAT($\mathcal{R}$)-instance α, then the expression would be a sum of products of the semiring values r that occur in the input ΣBF and the literals. We add for each occurrence r_i of r a variable v_r^i with $\alpha(v_r^i) = r, \alpha(\neg v_r^i) = e_\otimes$. Then we use the propositional theory T of the resulting AMC instance to specify which semiring values are included in the sum. □

3.5.5 Algebraic Measures

Recall that algebraic measures μ are given by a tuple $\langle \Pi, \alpha, \mathcal{R} \rangle$, where Π is a normal answer set program and α is a weighted QBF formula without quantifiers over the semiring $\mathcal{R}$. Note that in this chapter we restrict ourselves to ground programs Π. We are mostly interested in two reasoning tasks that involve algebraic measures, namely:

Problem:	AtomEval($\mathcal{R}$)
Input:	An algebraic measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$
Output:	The weight of a, i.e., $\mu(a) = \bigoplus_{\mathcal{I} \in \mathcal{AS}(\Pi), a \in \mathcal{I}} [\![\alpha]\!]_{\mathcal{R}}(\mathcal{I})$

Problem:	OverallEval($\mathcal{R}$)
Input:	An algebraic measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$
Output:	The overall weight, i.e., $\mu(\Pi) = \bigoplus_{\mathcal{I} \in \mathcal{AS}(\Pi)} [\![\alpha]\!]_{\mathcal{R}}(\mathcal{I})$

When comparing the evaluation of these queries with AMC and SAT($\mathcal{R}$), we see that algebraic measures are in a sense a combination of the strengths of both AMC and SAT($\mathcal{R}$). This is because α can be any weighted formula, instead of only a product of literal weights, and since Π can represent any logical theory over propositional variables.

Unsurprisingly, since both AMC and SAT($\mathcal{R}$) are NP($\mathcal{R}$)-complete we obtain the same for the reasoning tasks involving algebraic measures.

Theorem 96. *AtomEval($\mathcal{R}$) and OverallEval($\mathcal{R}$) are Karp s-reducible to SAT($\mathcal{R}$) and vice versa, and thus NP($\mathcal{R}$)-complete w.r.t. Karp s-reductions for every semiring $\mathcal{R}$.*

The proof uses the same ideas as that of Theorem 95 (see Appendix B.2.2 for the full proof).

3.5.6 Datalog Semiring Provenance

Green, Karvounarakis, and Tannen [GKT07] introduced a semiring-based semantics for relational algebra queries. Its main use case as described in the paper is to allow queries for the provenance of answers to standard queries. The semantics is capable of

expressing bag semantics, why-provenance and more. Here intuitively, bag semantics tell us not only *whether* we can derive a query but also *how often* we can derive it. Why-provenance tells us instead the different reasons *why* we have to derive a query. There are more possibilities but generally, the semiring semantics allows us to obtain more, often quantitative, information about the derivations of a query.

For positive logic programs, i.e. datalog programs, their semantics over a commutative semiring $(R, \oplus, \otimes, e_\oplus, e_\otimes)$ is as follows: the label of a query result $q(\overline{x})$ is the sum (using $\oplus$) of the labels of derivation trees for $q(\overline{x})$, where the label of a derivation tree is the product (using $\otimes$) of the labels of the leaf nodes (i.e. extensional atoms). As the number of derivation trees may be countably infinite, Green, Karvounarakis, and Tannen [GKT07] used ω-continuous semirings, where countable sums have a value. Examples of such semirings are $\mathbb{N}_\infty = (\mathbb{N} \cup \{\infty\}, +, \cdot, 0, 1)$, the natural number semiring extended with infinity, $\mathbb{B}$, and other idempotent semirings.

Example 32 (Bag Semantics)**.** *Consider the program*

$$r_1\colon q(X,Y) \leftarrow r(X,Y) \qquad\qquad r_2\colon q(X,Y) \leftarrow q(X,Z), q(Z,Y)$$

over $\mathbb{N}_\infty$ *and the semiring-weighted extensional database (edb)*

$$\{(r(a,b),2), (r(b,c),3), (r(b,a),4)\}.$$

Here, $\mathbb{N}_\infty$ *corresponds to bag semantics, which means that the label of a query* $q(\overline{x})$ *is the number of derivations of* $q(\overline{x})$*. The labels of the atoms in the semiring-weighted edb thus intuitively mean that there are* $2, 3$*, and* 4 *ways to derive* $r(a,b), r(b,c)$*, and* $r(b,a)$*, respectively.*

It follows that the label of $q(b,c)$ *under bag semantics is* 3 *due to the derivation* $q(b,c) \leftarrow r(b,c)$ *from* $r(b,c)$*, which itself has* 3 *derivations according to the edb, and the fact that this is the only derivation for* $q(b,c)$*. On the other hand, for* $q(a,a)$ *the label under bag semantics is* ∞*. This can be seen as follows: we can derive* $q(a,a)$ *from* $r(a,b), r(b,a)$ *in* 2×4 *ways according to their labels in the edb. But there exist infinitely many derivations of* $q(a,a)$ *from itself using rule* r_2 *instantiated as* $q(a,a) \leftarrow q(a,a), q(a,a)$*, leading to the label* ∞*.*

For the problem of computing the label of a query under provenance semantics it is not immediately clear how it relates to the other problems discussed in this section. Kimmig, Van den Broeck, and De Raedt [KVD17] touched upon the relation of this and other so called *Algebraic Derivation Counting* (ADC) formalisms to AMC. They include not only provenance queries but also semiring parsing [Goo99] and semiring-weighted dynamic programming [EGS05]. They showed that it is possible to reduce an AMC-instance T, α to ADC by computing all models of the theory T and viewing them as derivations. As they noted, this is obviously ineffective in general as the number of models of T can be exponential. The explicit computation of all the models is, however, not necessary for a reduction.

Theorem 97. *Given a positive, single-rule datalog program*

$$\Pi = \{q \leftarrow r_1(\vec{Y_1}), \ldots, r_n(\vec{Y_n})\}$$

and a semiring-weighted extensional database D over a commutative semiring $\mathcal{R}$ as input, it is NP($\mathcal{R}$)-complete w.r.t. Karp s-reductions to compute the label of q.

We assume here, as usual, that the database D is stored explicitly as a set of tuples $(r(\overline{X}), k)$, where r is a predicate and k is its semiring weight.

Proof. NP($\mathcal{R}$)-hardness: We provide a reduction from SumProd($\mathcal{R}$). Let $\mathcal{D}$ be a finite domain, $f_i : \mathcal{D}^{k_i} \to R, i = 1, \ldots, n$ be functions with the set of semiring values as co-domain, and let

$$\bigoplus_{X_1,\ldots,X_m \in \mathcal{D}} \bigotimes_{i=1}^{n} f_i(\vec{Y_i}), \tag{3.2}$$

be the SumProd($\mathcal{R}$)-instance at hand. Here, $\vec{Y_i}$ is a tuple consisting of k_i elements from $\{X_1, \ldots, X_m\}$, the set of variables of the SumProd($\mathcal{R}$)-instance, for $i = 1, \ldots, n$.

Consider the following positive, single-rule datalog program

$$\Pi = \{q \leftarrow r_1(\vec{Y_1}), \ldots, r_n(\vec{Y_n})\}$$

and the semiring-weighted extensional database

$$D = \bigcup_{i=1,\ldots,n} \{(r_i(\vec{y_i}), f_i(\vec{y_i})) \mid \vec{y_i} \in \mathcal{D}^{k_i}\}.$$

Both D and Π can be constructed in polynomial time from the SumProd($\mathcal{R}$)-instance. Furthermore, the label of the query q is equal to the result of evaluating the SumProd($\mathcal{R}$)-instance. Since SumProd($\mathcal{R}$) is NP($\mathcal{R}$)-complete w.r.t. Karp s-reductions, also provenance queries are NP($\mathcal{R}$)-hard.

NP($\mathcal{R}$)-membership: We can reduce computing the label of q for a provenance instance to a SumProd($\mathcal{R}$)-instance analogously to obtain that provenance queries of the above form over $\mathcal{R}$ are in NP($\mathcal{R}$). This implies NP($\mathcal{R}$)-completeness overall. □

Note that we only showed NP($\mathcal{R}$)-completeness for a restricted case of provenance computations here, where the datalog program has only one rule. In fact, it is not possible to evaluate provenance queries in NP($\mathcal{R}$) in general. This is not because of hard special cases caused by the possibility of countably infinite sums, as already Green, Karvounarakis, and Tannen [GKT07] gave an algorithm to recognize this case. The hardness comes from higher expressivity in two dimensions instead. On the one hand, in the Boolean setting, it is known that the evaluation of a datalog program is EXPTIME-complete, even for a fixed extensional database D [Dan+01]. Additionally, when the extensional database is not fixed evaluating Boolean queries defined by a single rule is

also already EXPTIME-complete [GP03]. Thus, it seems unlikely that we can stay in $\mathrm{NP}(\mathcal{R})$, even when $\mathcal{R} = \mathbb{B}$.

For other semirings, provenance queries are even provably more expressive than ΣBFs, already for ground programs Π, i.e., programs where only nullary predicates are used. The latter allow us to express circuit-like terms.

Example 33. *Let $n \in \mathbb{N}$ and consider the positive datalog program*

$$\Pi = \{a_{i+1} \leftarrow a_i, a_i \mid i = 1, \ldots, n-1\}$$

and the semiring-weighted extensional database $D = \{(a_1, 2)\}$. Then the label l_i of a_i is given by the recurrence relation $l_1 = 2$ and $l_{i+1} = a_i^2$ for $i > 0$. Therefore, the label of a_n is $(\ldots(2^2)^2\ldots)^2 = 2^{2^n}$, which is exponential in the size of the input.

3.5.7 Semiring-induced Propositional Logic

Larrosa, Oliveras, and Rodrıǵuez-Carbonell [LOR10] introduced an extension of propositional logic with a semiring-based semantics. They confined to CNF formulas and associated a weight from the semiring with each of the clauses. Formally:

Definition 98 (SRPL)**.** *Given a* semiring-labeled CNF $F = \{(C_1, w_1)\ldots, (C_n, w_n)\}$ *over a commutative semiring $\mathcal{R}$, where each C_i is a clause and $w_i \in R$ is a value from $\mathcal{R}$, the semantics $\phi_i(\mathcal{I})$ of clause C_i under interpretation $\mathcal{I}$ is*

$$\phi_i(\mathcal{I}) = \begin{cases} w_i & \text{if } \mathcal{I} \not\models C_i \\ e_\otimes & \text{otherwise.} \end{cases}.$$

Here, $\models$ is the satisfaction relation of classical propositional logic.

Furthermore, $\phi_F(\mathcal{I})$, the semantics of F under interpretation $\mathcal{I}$ is given by

$$\phi_F(\mathcal{I}) = \bigotimes_{i=1}^{n} \phi_i(\mathcal{I}).$$

This is reminiscent of the evaluation of a SumProd($\mathcal{R}$)-instance for one assignment $\vec{x}$ to the variables $\vec{X}$: both the term under the assignment $\vec{x}$ and $\phi_F(\mathcal{I})$ are products of values functionally determined by $\vec{x}$ and $\mathcal{I}$, respectively.

As with SumProd($\mathcal{R}$), one is not interested in the value for a single assignment but in the sum over all assignments, called *marginal* here, which is defined as follows:

Definition 99 (Marginalization Problem)**.** *Given a semiring-labeled CNF F over a commutative semiring $\mathcal{R}$ as input, the* marginalization problem over $\mathcal{R}$ *is to compute*

$$mrg(F) = \bigoplus_{\mathcal{I} \in Int(F)} \phi_F(\mathcal{I}),$$

where $Int(F)$ denotes for a semiring-labeled CNF F the set of all interpretations of variables in F.

Larrosa, Oliveras, and Rodrıǵuez-Carbonell [LOR10] noted that many typical problems, like SAT, #SAT and MAX-SAT can be expressed as marginalization problems over different semirings. We show that moreover the marginalization problem is NP($\mathcal{R}$)-complete.

Theorem 100. *The marginalization problem over $\mathcal{R}$ is NP($\mathcal{R}$)-complete w.r.t. Karp s-reductions for every commutative semiring $\mathcal{R}$.*

Proof (Sketch, see Appendix B.2.2 for the full proof). NP($\mathcal{R}$)-membership is easy to establish by a simple guess and evaluate algorithm.

We can prove NP($\mathcal{R}$)-hardness by a reduction from AMC over $\mathcal{R}$. W.l.o.g. we assume that the propositional theory T of the given AMC instance is a CNF with clauses $C_1, \ldots, C_n$ and the labeling function is α. Then the result of the marginalization problem for

$$F = \{(C_1, e_\oplus), \ldots, (C_n, e_\oplus)\} \cup \{(\neg v, \alpha(v)) \mid v \in V\} \cup \{(v, \alpha(\neg v)) \mid v \in V\},$$

where V is the set of variables of T, is the algebraic model count of T with respect to α. □

3.5.8 Other Frameworks

There are also other frameworks, such as algebraic Prolog [KVD11], and semiring induced valuation algebras [KW08] that we can show to be NP($\mathcal{R}$)-complete using our methodology and the previous reductions. We refrain from doing so, since the corresponding evaluation problems either already come with reductions to AMC [KVD17] or are very similar to other frameworks as in the case of semiring induced valuation algebras and SCSPs.

3.6 Relation to Well-Known Complexity Classes

We were able to show in the previous section that NP($\mathcal{R}$) can be used to characterize the complexity of many different semiring formalisms that each in a way use the guess-evaluate-aggregate pattern. However, these results only characterize the complexity over different semirings on a relatively abstract level by using SRTMs as a machine model. In order to gain more *tangible* insights into their complexity in a usual setting, we relate NP($\mathcal{R}$) to well-known classical complexity classes defined on Turing machines in this section.

For this purpose, we must encode semiring values in a finite tape alphabet. As we will see, the choice of encoding can strongly influence the complexity of computations over a semiring. To address this, we introduce conditions for *efficient* encodings that we can use to limit the influence of the encoding.

Equipped with this, we first link well-known complexity classes such as NP, #P, and OptP with NP($\mathcal{R}$) for specific semirings, thus showing that NP($\mathcal{R}$) is in a sense a proper generalization of existing quantitative complexity classes.

We then turn to the more general case of commutative semirings as a whole and subclasses thereof. Here, we prove on the one hand general lower bounds, upper bounds and a tetrachotomy result that separates the non-trivial semirings into four distinct types of problems each of which is strongly connected to either NP, Mod_pP, $\text{NP} \cup \text{Mod}_p\text{P}$ or #P.

3.6.1 Encoding Semirings

To represent semiring values on classical Turing machines, we need to encode their values using an *encoding function*. We introduce the necessary notions and consider the implications that the choice of the encoding has on the complexity.

Definition 101 (Encoding Function, Encoded Semiring)**.** *Let $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ be a semiring. Then an injective function $e : R \to \{0,1\}^*$ is an* encoding function*.*

We let $e(\mathcal{R}) = (e(R), \oplus, \otimes, e(e_\oplus), e(e_\otimes))$ denote the encoded semiring *given by $e(R) = \{e(r) \mid r \in R\}$ and $\odot$ on $e(R)$, s.t. $e(r_1) \odot e(r_2) = e(r_1 \odot r_2)$ for $\odot = \oplus, \otimes$.*

Given an encoded value $e(r)$ we define $\|r\|_e$, the size of r w.r.t. e*, as the length of the bitstring $e(r)$, i.e. $|e(r)|$.*

Now, we can use classical machines to solve $\text{SAT}(e(\mathcal{R}))$ and consider its complexity in terms of classical complexity classes. Naturally, it depends on the complexity of addition and multiplication. While for $\mathbb{N}$ and $\mathbb{B}$ these operations are "easy", i.e., feasible in polynomial time given a binary encoding, this is not the case for arbitrary semirings. A single multiplication may even be undecidable.

Example 34 (Arbitrary Complexity Semirings)**.** *Given $M \subseteq \{0,1\}^*$ and $\succ$, the lexicographical order on $\{0,1\}^*$, we define the semiring $\mathcal{R}_M = (\{0,1\}^* \cup \{\mathbf{0}, \mathbf{1}\}, \max_\succ, \otimes, \mathbf{0}, \mathbf{1})$, where $\mathbf{1} \succ m \succ \mathbf{0}$ for $m \in \{0,1\}^*$ and*

$$m_1 \otimes m_2 := \begin{cases} \min_\succ(m_1, m_2) & m_1, m_2 \in M \cup \{\mathbf{0}\} = S \\ m_1 & m_1 \in S, m_2 \notin S \\ m_2 & m_2 \in S, m_1 \notin S \\ \min_\succ(m_1, m_2) & \text{otherwise.} \end{cases}$$

Then multiplication requires deciding $m_i \in M$. When M corresponds to the Halting problem, we have undecidability.

However, the difficulty stems from the encoding.

Example 35 (Example 34 cont.)**.** *If the encoding e maps $m \in \{0,1\}^*$ to $(m,1)$ if $m \in M$ and to $(m,0)$ if $m \notin M$, then multiplication and addition in $e(\mathcal{R}_M)$ are computable in linear time.*

Our intuition is that there are two sources of complexity. One is the encoding and the other seems to be the amount of information that the weighted semantics gives us about the formula. The latter is determined by the non-collapsing terms in the semiring. For illustration purposes, consider that over $\mathbb{N}[(x_i)_k]$ the coefficients c_1, c_2 are retained by the sum $c_1x_1 + c_2x_2$, over $\mathbb{N}$ only the sum $c_1 + c_2$ is retained after addition, and over $\mathbb{B}$ the value $c_1 \vee c_2$ only tells us if at least one of the values was 1. As a consequence SAT($\mathbb{N}[(x_i)_k]$)-instances are at least as hard as SAT($\mathbb{N}$)-instances, since $\mathbb{N}$ is a subset of $\mathbb{N}[(x_i)_k]$ with the same addition and multiplication on $\mathbb{N}$. Additionally, SAT($\mathbb{N}$)-instances seem to be strictly harder than SAT($\mathbb{B}$)-instances as NP can easily be reduced to #P.

We focus on the second source of complexity and address the first by introducing the notion of an efficient encoding.[3]

Definition 102 (Efficiently Encoded Semiring)**.** *Let $e(\mathcal{R})$ be an encoded semiring. Then $e(\mathcal{R})$ is* efficiently encoded *if there exists a polynomial $p(x)$ s.t. for all $e(r_1), \ldots, e(r_n) \in e(R)$ it holds that*

1. $\|\bigotimes_{i=1}^{n} r_i\|_e \leq p(n) \max_{i=1,\ldots,n} \|r_i\|_e$,
2. $\|\bigoplus_{i=1}^{n} r_i\|_e \leq p(\log_2(n)) \max_{i=1,\ldots,n} \|r_i\|_e$, *and*
3. $e(r), e(r') \mapsto e(r \odot r')$ *is in FP for* $\odot = \oplus, \otimes$.

Conditions 1) and 2) ensure that successive multiplications resp. additions do not cause space explosion, even for sums with exponentially many terms. Condition 3) is necessary since we at least need single additions and multiplications to be tractable if we want to solve problems over a semiring efficiently. The idea behind these conditions is to separate encodings that behave "efficiently" both with respect to space and time and those that do not. For this, we use restrictions that mirror and slightly relax the properties that the prototypical binary encoding "bin(.)" of integers satisfies.

As desired and expected, for integers, a binary representation satisfies the restrictions, whereas a unary representation does not.

Example 36. *With binary representation $bin(n) = b_0 \ldots b_m$ s.t. $n = \sum_{i=1}^{m} b_i 2^i$, the semiring $\mathbb{N}$ of the natural numbers is efficiently encoded. With the unary representation $unary(n)$, which represents a natural number as a string of n characters, the natural numbers are* not *efficiently encoded and in fact do not satisfy any of the conditions 1)-3).*

Furthermore, also $\mathbb{N}[(x_i)_\infty]$ is not efficiently encoded when polynomials are encoded as lists of monomials and every monomial of the form $c_{\vec{i}} x_{i_1}^{e_{i_1}} \ldots x_{i_n}^{e_{i_n}}$ is encoded using a unary or binary representation of the coefficients $c_{\vec{i}}$, the exponents e_{i_j}, and variable indices i_j.

The conditions of Definition 102 are mild in practice, as besides $\mathbb{N}$ many common semirings, e.g. $\mathbb{Z}, \mathbb{Q}$, and $\mathcal{R}_{\max,+}$, are efficiently encodable. They remain so under sharpenings like $p(n) = \mathcal{O}(n)$, but this may lead to less "natural" encodings.

[3] Compared to Definition 36 we restrict ourselves to the operations that exist in every semiring, here.

When a semiring is efficiently encoded, we can establish a polynomial space upper bound on the complexity of ΣBF evaluation.

Proposition 103 (FPSpace(poly) Upper-Bound)**.** *If $e(\mathcal{R})$ is an efficiently encoded commutative semiring, then SAT($e(\mathcal{R})$) is in FPSpace(poly).*

Proof (Sketch, see Appendix B.3.1 for the full proof). As we have seen before, the value of an SAT($e(\mathcal{R})$)-instance α can be expressed as a sum of products of the semiring values in α by implicitly applying the distributive law on α during evaluation as in Algorithm 1. The length of the encoding of each term of this sum is polynomially bounded in the size of α due to condition 1 of efficient encodedness. Since there are only single exponentially many such addends, it follows from condition 2 of efficient encodedness that the final result also has polynomial size in the size of the biggest term and is therefore also polynomial in the size of the input. By generating the terms one after the other we, thus, stay in polynomial space. □

3.6.2 Results for Specific Semirings

Before we consider the complexity in dependence on the semiring parameter in general, we relate well-known classical complexity classes to NP($e(\mathcal{R})$) by showing that they share SAT($e(\mathcal{R})$) as a complete problem for different specific semirings.

Theorem 104. *For $(\mathcal{R}, \mathcal{C}) = (\mathbb{B}, NP)$, $(\mathbb{N}, \#P)$, $(\mathbb{Z}, GapP)$, $(\mathcal{R}_{\max,+}, OptP)$ and the binary representation bin of the integers, SAT(bin($\mathcal{R}$)) is $\mathcal{C}$-complete w.r.t. Karp reductions.*

Proof (Sketch, see Appendix B.3.2 for the full proof). Membership holds as bin(n) satisfies Definition 102 and we can, thus, interpret Algorithm 1 as a non-deterministic Turing machine algorithm that generates the correct polynomial size outputs. For hardness we use reductions from SAT, #SAT, computing the permanent of an integer matrix, and LexMaxSAT, respectively. □

Note that even though every function in OptP can be reduced to a SAT(bin($\mathcal{R}_{\max,+}$)), there are functions in OptP that cannot be computed in NP($\mathcal{R}_{\max,+}$) (or NP(bin($\mathcal{R}_{\max,+}$)) for that matter), e.g., $x \mapsto 2^{|x|}$. The problem here is that SRTMs, contrary to classical Turing Machines, cannot interpret bit-strings as the encodings of semiring values but always store semiring values in a single cell on the tape. Therefore, if we first apply a classical Karp reduction we can interpret bit-strings as the encodings of semiring values and, thus, may give us more power than applying a Karp s-reduction. Without the power of interpreting bit-strings, SRTMs can only generate semiring values by multiplying numbers from a finite set R' or the input. The fact that multiplication in $\mathcal{R}_{\max,+}$ is $+$ implies that we can only generate numbers that are polynomial in the numbers in the input and R'. For NP,#P and GapP this effect does not occur.

3.6.3 Results for Classes of Semirings

Apart from completeness results for specific semirings, we also care about intuition on why some semirings come with a higher complexity than others, and about results that help to characterize new semirings based on their properties. Thus, we consider the complexity of different classes of semirings that satisfy some property, which allows us to make non-trivial claims about the complexity entailed by evaluating problems over them.

General Lower Bound and Tetrachotomy

First, we consider how hard SAT($\mathcal{R}$) has to be at least in general. For this we show the following result:

Theorem 105 (General Lower Bound)**.** *Let $e(\mathcal{R})$ be an encoded commutative semiring. Then one of the following holds:*

1. *$e(\mathcal{R}) = \mathbb{T}$, i.e., the semiring is trivial*
2. *SAT$(e(\mathcal{R}))$ is Mod$_p$P-hard with respect to counting reductions for some $p \in \mathbb{N}$ or*
3. *SAT$(e(\mathcal{R}))$ is NP-hard with respect to counting reductions.*

This means that the problem is either trivial (case 1), or expected not to be solvable in FP under common complexity theoretic assumptions such as the Exponential Time Hypothesis [IP01], otherwise.

Before we continue with the proof, we first make the following remark that we will use throughout the rest of this chapter. Let $C = l_1 \vee l_2 \vee l_3$ be a clause and let $\overline{l}$ for a literal l denote the opposite of l, i.e., $\overline{a} = \neg a$ and $\overline{\neg a} = a$ for any propositional variable a. Then:

Lemma 106. *For every non-trivial semiring $\mathcal{R}$, an interpretation $\mathcal{I}$ satisfies C iff for*

$$d(C) := l_1 + \overline{l_1} * l_2 + \overline{l_1} * \overline{l_2} * l_3$$

it holds that $[\![d(C)]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\otimes}$. Furthermore, if $\mathcal{I}$ does not satisfy C, then $[\![d(C)]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\oplus}$.

Proof of Lemma 106. We proceed by case distinction. Let $\mathcal{I}$ be an interpretation.

Case $\mathcal{I} \models l_1$: Then

$$[\![d(C)]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\otimes} \oplus e_{\oplus} \oplus e_{\oplus} = e_{\otimes}$$

since $[\![l_1]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\otimes}$ and $[\![\overline{l_1}]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\oplus}$.

Case $\mathcal{I} \not\models l_1$, $\mathcal{I} \models l_2$: Then

$$[\![d(C)]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\oplus} \oplus e_{\otimes} \oplus e_{\oplus} = e_{\otimes}.$$

Case $\mathcal{I} \not\models l_1$, $\mathcal{I} \not\models l_2$, $\mathcal{I} \models l_3$: Then

$$[\![d(C)]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\oplus} \oplus e_{\oplus} \oplus e_{\otimes} = e_{\otimes}.$$

Case $\mathcal{I} \not\models l_1$, $\mathcal{I} \not\models l_2$, $\mathcal{I} \not\models l_3$: Then

$$[\![d(C)]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\oplus} \oplus e_{\oplus} \oplus e_{\oplus} = e_{\oplus}.$$

This covers all cases and we see that when $\mathcal{I} \models C$, then $[\![d(C)]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\otimes}$ and otherwise $[\![d(C)]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\oplus}$. □

Furthermore, for $n \in \mathbb{N}$ and $r \in R$, we let

$$n \cdot r = \begin{cases} e_{\oplus} & n = 0 \\ ((n-1) \cdot r) \oplus r & n > 0 \end{cases}.$$

Proof of Theorem 105. Assume $e(\mathcal{R}) \neq \mathbb{T}$, since otherwise we are done.

We distinguish the following cases. If for all $n \in \mathbb{N}$ it holds that $n \cdot e(e_{\otimes}) = e(e_{\oplus})$ implies $n = 0$, then $\mathrm{SAT}(e(\mathcal{R}))$ is NP-hard with respect to counting reductions. For the proof, consider a 3CNF $\phi = \bigwedge_{i=1}^{n} C_i$ with variables $v_1, \ldots, v_m$. Then ϕ is satisfiable iff for $\alpha = \Sigma v_1 \ldots \Sigma v_m \Pi_{i=1}^{n} d(C_i)$ it holds that $[\![\alpha]\!]_{e(\mathcal{R})}(\emptyset) \neq e(e_{\oplus})$.

Otherwise, let $M = \langle e(e_{\otimes}) \rangle$ and

$$p = \min\{n \in \mathbb{N} \mid n > 0, n \cdot e(e_{\otimes}) = e_{\oplus}\}.$$

We claim that $M = \{n \cdot e(e_{\otimes}) \mid n \in \mathbb{N}\}$. This can be seen as follows. By definition of generated semirings, $\langle e(e_{\otimes}) \rangle$ is the smallest semiring with addition $\oplus$ and multiplication $\otimes$ that contains $e(e_{\otimes})$. We go over the relevant semiring axioms one by one. First, $e(e_{\oplus}) = 0 \cdot e(e_{\otimes})$ and thus of the desired form. Second, $e(e_{\otimes}) = 1 \cdot e(e_{\otimes})$ and thus of the desired form. Next, M must be closed under $\oplus$. So let $n \cdot e(e_{\otimes}), m \cdot e(e_{\otimes}) \in M$. Then,

$$(n \cdot e(e_{\otimes})) \oplus (m \cdot e(e_{\otimes})) = ((n+1) \cdot e(e_{\otimes})) \oplus ((m-1) \cdot e(e_{\otimes})) = \cdots = (n+m) \cdot e(e_{\otimes}),$$

since $\oplus$ is commutative. Last but not least, M must be closed under $\otimes$. So let $n \cdot e(e_{\otimes}), m \cdot e(e_{\otimes}) \in M$. Then,

$$\begin{aligned} (n \cdot e(e_{\otimes})) \otimes (m \cdot e(e_{\otimes})) &= (n \cdot e(e_{\otimes})) \otimes ((m-1) \cdot e(e_{\otimes})) \oplus (n \cdot e(e_{\otimes})) \otimes e(e_{\otimes}) \\ &= \ldots \\ &= (n \cdot m) \cdot e(e_{\otimes}) \otimes e(e_{\otimes}) \\ &= (n \cdot m) \cdot e(e_{\otimes}), \end{aligned}$$

since $\otimes$ distributes over $\oplus$ and $e(e_{\otimes}) \otimes e(e_{\otimes}) = e(e_{\otimes} \otimes e_{\otimes}) = e(e_{\otimes})$. This proves the claim.

Furthermore, for $n \geq p$ it holds that $n \cdot e(e_\otimes) = (n-p) \cdot e(e_\otimes)$. Therefore,

$$M = \{e(e_\oplus), 1 \cdot e(e_\otimes), \ldots, (p-1) \cdot e(e_\otimes)\} \equiv \mathbb{Z}_p.$$

It follows that $\mathrm{SAT}(e(\mathcal{R}))$ is $\mathrm{MOD}_p\mathrm{P}$-hard with respect to counting reductions. We give a reduction from $\mathrm{MOD}_p\mathrm{3CNF}$, which is the problem of deciding whether the number of models N that a given 3CNF $\phi = \bigwedge_{i=1}^{n} C_i$ with variables $v_1, \ldots, v_m$ has is not equal to zero modulo p; this problem is $\mathrm{MOD}_p\mathrm{P}$-complete [BG92]. For the proof of hardness, we again take $\alpha = \Sigma v_1 \ldots \Sigma v_m \Pi_{i=1}^{n} d(C_i)$ and observe that $[\![\alpha]\!]_{e(\mathcal{R})}(\emptyset) = n^* \cdot e(e_\otimes)$, where $n^* \equiv N \mod p$. Since the cardinality of M is finite, we can decide in constant time what the unique value $n^* \in \{0, \ldots, p-1\}$ is. The claim follows. □

If we restrict ourselves to the case where we cannot have weights and consider an efficiently encoded commutative semiring, we obtain a tetrachotomy result. This means, we split the complexity associated with a commutative semiring into four distinct cases depending on its properties. The properties we use are periodicity and offset of a periodic semiring.

Definition 107 (Periodic Semiring,Periodicity, Offset)**.** *A semiring $\mathcal{R}$ is* periodic *if for all $r \in R$ there exist $n, m \in \mathbb{N}$ such that $\bigoplus_{i=1}^{n} r = \bigoplus_{i=1}^{m} r$.*

Let $\mathcal{R}$ be a periodic semiring. Then the periodicity *of $\mathcal{R}$ is the smallest number $p > 0$ such that some $o \geq 0$ with $o \cdot e_\otimes = (o+p) \cdot e_\otimes$ exists. Let p be the periodicity of $\mathcal{R}$. Then the* offset *of $\mathcal{R}$ is the smallest number $o \geq 0$ such that $o \cdot e_\otimes = (o+p)e_\otimes$.*

Both periodicity and offset are well defined, i.e., there are unique numbers p, o that satisfy the definition for periodic semirings.

Note that a periodicity of p does not imply that $(p+1) \cdot e_\otimes = e_\otimes$.

Example 37. *Consider the semiring*

$$\mathbb{N}_{\leq o} = (\{0, 1, \ldots, o\}, +, \cdot, 0, 1),$$

where $i + j := \min(o, i+j)$ and $i \cdot j := \min(o, i \cdot j)$, i.e., the semiring over the natural numbers less than or equal to o. $\mathbb{N}_{\leq o}$ is periodic with periodicity 1 but $1 + 1 \neq 1$ when $o > 1$.

Corollary 108. *Let $\mathcal{R}$ be a periodic semiring with periodicity p and offset o. Then for all $r \in R$ and $o' \geq o$ it holds that $o' \cdot r = (o' + p) \cdot r$.*

Proof. This can be shown by exploiting that $e_\otimes$ is the neutral element for multiplication and the distributive law:

$$\begin{aligned}\bigoplus_{i=1}^{o'} r = \bigoplus_{i=1}^{o'} r \otimes e_\otimes = r \otimes \bigoplus_{i=1}^{o'} e_\otimes &= r \otimes \left(\bigoplus_{i=1}^{o'-o} e_\otimes \oplus \bigoplus_{i=1}^{o} e_\otimes\right) \\ &= r \otimes \left(\bigoplus_{i=1}^{o'-o} e_\otimes \oplus \bigoplus_{i=1}^{o+p} e_\otimes\right) = r \otimes \bigoplus_{i=1}^{o'+p} e_\otimes = \bigoplus_{i=1}^{o'+p} r.\end{aligned}$$

□

With this in mind, we can show that the complexity of evaluating $\mathrm{SAT}(e(\langle e_\otimes \rangle_{e(\mathcal{R})}))$-instances, i.e., instances over some efficiently encoded semiring $e(\mathcal{R})$ that only use weights of the form $k \cdot e(e_\otimes)$, falls into one of four distinct cases. Furthermore, we show that the category a semiring falls into is determined by whether it is periodic, and if so, by its periodicity and offset.

Theorem 109 (Tetrachotomy)**.** *Let $e(\mathcal{R}) \neq \mathbb{T}$ be an efficiently encoded commutative semiring. Then we have exactly one of the four following situations:*

- *$e(\mathcal{R})$ is periodic with periodicity 1 and $SAT(e(\langle e_\otimes \rangle))$ is NP-hard w.r.t. counting reductions and in $FP^{NP[\mathcal{O}(1)]}$;*
- *$e(\mathcal{R})$ is periodic with periodicity $p \geq 2$ and offset 0 and $SAT(e(\langle e_\otimes \rangle))$ is $\mathrm{Mod}_p\mathrm{P}$-hard w.r.t. counting reductions and in $FP^{\mathrm{Mod}_p P[\mathcal{O}(1)]}$;*
- *$e(\mathcal{R})$ is periodic with periodicity $p \geq 2$ and offset $o > 0$ and $SAT(e(\langle e_\otimes \rangle))$ is NP-hard, $\mathrm{Mod}_p\mathrm{P}$-hard w.r.t. counting reductions and in $FP^{(NP \cup \mathrm{Mod}_p P)[\mathcal{O}(1)]}$;*
- *$e(\mathcal{R})$ is not periodic and #SAT is in $FNP^{SAT(e(\langle e_\otimes \rangle))[1]}$. Furthermore, if additionally there is a polynomial p such that for all $k \geq 0$ it holds that $2^{p(\|k \cdot e_\otimes\|)} \geq k$, then $SAT(e(\langle e_\otimes \rangle))$ is in $FP^{\#P[1]}$.*

Note that practically for efficient evaluation also parallel oracle calls to NP and $\mathrm{Mod}_p\mathrm{P}$ could be used.

This result shows that if we only consider instances without semiring values, the already known complexity classes NP, $\mathrm{Mod}_p\mathrm{P}$ and #P suffice to bound the complexity in a reasonable fashion, even if we allow general commutative countable semirings. Unfortunately, the next subsection shows that this does not simply generalize to the case when we have non-trivial weights.

We also would like to draw special attention to the last case, where $e(\mathcal{R})$ is not periodic. Here, the bounds we can give are weaker than in the other three cases. This is because if a semiring is periodic, then $e(\langle e_\otimes \rangle)$ is finite, which implies that we can compute for any $r \in e(\langle e_\otimes \rangle)$ in constant time a value $k \in \mathbb{N}$ such that $r = k \cdot e_\otimes$. On the other hand, if $e(\mathcal{R})$ is not periodic, it is not clear whether given $r \in e(\langle e_\otimes \rangle)$ it is possible even in polynomial time to compute $k \in \mathbb{N}$ such that $r = k \cdot e_\otimes$. The restriction that $e(\mathcal{R})$ is efficiently encoded helps us, since it allows us to check in polynomial time whether $r = k \cdot e_\otimes$, given $e(r)$ and k. However, as described by Sharma and Singh [SS16], it is an open problem whether easy to compute functions (like $f(k) = k \cdot e_\otimes$) have an easy to compute inverse (like $f^{-1}(e(r))$). This explains the weaker lower bound on the hardness of $\mathrm{SAT}(e(\langle e_\otimes \rangle))$.

For the upper bound, our proof for the reduction of $\mathrm{SAT}(e(\langle e_\otimes \rangle))$ to #SAT requires that for all $r \in e(\langle e_\otimes \rangle)$ it holds that the value $k \in \mathbb{N}$ such that $r = k \cdot e_\otimes$ is single exponential in the size of the encoding $\|r\|_e$. We leave the question of the existence of an efficiently encoded semiring, where this does not hold open.

Proof. Clearly, $e(\mathcal{R})$ is exactly one of

1. periodic with periodicity 1
2. periodic with periodicity $p \geq 2$ and offset 0
3. periodic with periodicity $p \geq 2$ and offset $o > 0$
4. not periodic

What is left to show is the corresponding claim for the complexity of $\mathrm{SAT}(e(\langle e_\otimes \rangle))$.

Case 1. So let $e(\mathcal{R})$ have periodicity 1 and let $\alpha = \Sigma v_1 \dots \Sigma v_l \gamma$, where γ is quantifier-free, be a $\mathrm{SAT}(e(\langle e_\otimes \rangle))$-instance. Since γ only contains weights from $e(\langle e_\otimes \rangle)$, we know that for any interpretation $\mathcal{I}$ the semantics of γ is $m \cdot e(e_\otimes)$ for some $m \geq 0$. Further, let o be the offset of $e(\mathcal{R})$. Then we know that $(o+m) \cdot e(e_\otimes) = o \cdot e(e_\otimes)$ for all $m \geq 0$. Thus, $[\![\gamma]\!]_{e(\mathcal{R})}(\mathcal{I}) \in \{0 \cdot e(e_\otimes), \dots, o \cdot e(e_\otimes)\}$ and also $[\![\alpha]\!]_{e(\mathcal{R})}(\emptyset) \in \{0 \cdot e(e_\otimes), \dots, o \cdot e(e_\otimes)\}$.

Thus, we can use o^2 queries to an NP-oracle to determine for each value $i = 1, \dots, o$ whether there are $1, \dots, o$ or more interpretations with value $i \cdot e(e_\otimes)$. From this we can derive the value of α. Thus, $\mathrm{SAT}(e(\langle e_\otimes \rangle))$ is in $\mathrm{FP}^{\mathrm{NP}[\mathcal{O}(1)]}$, since o only depends on the semiring and not on the instance. For NP-hardness we can use the same construction as in the proof of Theorem 105.

Case 2. So let $e(\mathcal{R})$ have periodicity $p \geq 2$ and offset 0 and let $\alpha = \Sigma v_1 \dots \Sigma v_l \gamma$, where γ is quantifier-free, be a $\mathrm{SAT}(e(\langle e_\otimes \rangle))$-instance. Since γ only contains weights from $e(\langle e_\otimes \rangle)$, we know that for any interpretation $\mathcal{I}$ the semantics of α is $m \cdot e(e_\otimes)$ for some $0 \leq m < p$.

Now, let $f \in \#\mathrm{P}$ be a function that evaluates $\mathrm{SAT}(\mathrm{bin}(\mathbb{N}))$-instances. We can use f on α by replacing every value $e(k \cdot e_\otimes)$ by $\mathrm{bin}(k)$. Since $k \equiv 0 \mod p$ iff $k \cdot e(e_\otimes) = 0 \cdot e(e_\otimes)$, checking whether the semantics of α is $0 \cdot e(e_\otimes)$ is in $\mathrm{Mod}_p\mathrm{P}$ by definition.

Additionally, since $f \in \#\mathrm{P}$ implies $f + i \in \#\mathrm{P}$, we can use p queries to a $\mathrm{Mod}_p\mathrm{P}$-oracle to determine for each value $i = 1, \dots, p$ whether the semantics of α is $i \cdot e(e_\otimes)$. Thus, $\mathrm{SAT}(e(\langle e_\otimes \rangle))$ is in $\mathrm{FP}^{\mathrm{Mod}_p\mathrm{P}[\mathcal{O}(1)]}$, since p only depends on the semiring and not on the instance. For $\mathrm{Mod}_p\mathrm{P}$-hardness we can use the same construction as in the proof of Theorem 105.

Case 3. So let $e(\mathcal{R})$ have periodicity $p \geq 2$ and offset $o > 0$ and let $\alpha = \Sigma v_1 \dots \Sigma v_l \gamma$, where γ is quantifier-free, be a $\mathrm{SAT}(e(\langle e_\otimes \rangle))$-instance. Since γ only contains weights from $e(\langle e_\otimes \rangle)$, we know that for any interpretation $\mathcal{I}$ the semantics of α is $i \cdot e(e_\otimes)$ for some $0 \leq i < o + p$.

The rest is a combination of case 1. and 2.: we first check whether $i < o$ using o^2 calls to an NP-oracle. If $i < o$ we are done, otherwise, we use p calls to a $\mathrm{Mod}_p\mathrm{P}$-oracle to

obtain the value i. For hardness we again use the same construction as in the proof of Theorem 105.

Case 4. So let $e(\mathcal{R})$ not be periodic and let $\alpha = \Sigma v_1 \dots \Sigma v_l \gamma$, where γ is quantifier-free, be a SAT($e(\langle e_\otimes \rangle)$)-instance. Since γ only contains weights from $e(\langle e_\otimes \rangle)$, we know that for any interpretation $\mathcal{I}$ the semantics of α is $m \cdot e(e_\otimes)$ for some $m \in \mathbb{N}$. If we can find out for each r that occurs in γ the natural number k_r such that $r = k_r \cdot e_\otimes$, we can reduce the evaluation of α to evaluating α over bin($\mathbb{N}$), as this will have result m. Given m we can then compute $m \cdot e(e_\otimes)$ in polynomial time. This establishes the existence of a counting reduction from SAT($e(\langle e_\otimes \rangle)$) to #SAT, if we can compute given r the natural number k_r such that $r = k_r \cdot e_\otimes$.

If we know that for $k \cdot e(e_\otimes)$ it holds that $2^{p(\|k \cdot e_\otimes\|)} \geq k$ for some polynomial p, then, given the encoding $e(r)$ we can guess all numbers $0 \leq k \leq 2^{p(\|r\|_e)}$ and check whether $k \cdot e(e_\otimes) = e(r)$ in non-deterministic polynomial time. By doing this for all weights $e(r)$ that occur in α before evaluating α non-deterministically over $\mathbb{N}$ and rejecting if a guess was wrong, we can reduce SAT($e(\langle e_\otimes \rangle)$) to #SAT.

As for the containment of #SAT in FNP$^{\text{SAT}(e(\langle e_\otimes \rangle))[1]}$, consider the following algorithm: given a #SAT-instance ϕ construct a SAT($e(\langle e_\otimes \rangle)$)-instance α such that the semantics of α is $m \cdot e(e_\otimes)$, where m is the number of models of ϕ. This is possible in polynomial time by first constructing a 3CNF $\psi = \bigwedge_{i=1}^{n} C_i$ with variables $v_1, \dots, v_m$ and the same number of models and using $\alpha = \Sigma v_1 \dots \Sigma v_m \Pi_{i=1}^{n} d(C_i)$, where $d(C_i)$ is as in Lemma 106. Then, evaluate α, guess $0 \leq n \leq 2^{|Vars(\phi)|}$, where $Vars(\phi)$ denotes the set of variables in ϕ, and compute $n \cdot e(e_\otimes)$. If $n \cdot e(e_\otimes)$ is equal to the semantics of α accept and return n, otherwise reject. □

Upper Bounds using Polynomial Semirings

We have seen results on the hardness of commutative semirings in general and some containment results for the case when we only use weights of the form $k \cdot e_\otimes$ for some $k \in \mathbb{N}$ in the evaluation of an instance. In order to also obtain containment results for the case when we do have arbitrary weights, we need to apply a different kind of strategy.

The problem with proper weights is that they intuitively allow us to preserve more information when they are added or multiplied. We start with a formalization of this intuition by showing that *epimorphisms*, which can be seen to map a semiring $\mathcal{R}_1$ to an at most as informative semiring $\mathcal{R}_2$, can be used to reduce SAT($\mathcal{R}_2$) to SAT($\mathcal{R}_1$).

This makes polynomial semirings such as $\mathbb{N}[(x_i)_\infty]$ very interesting for us since they are, as we will see, the most information preserving countable commutative semirings and can thus always be reduced to. We show that in some cases this strategy is unlikely to work but also prove that there are still large subclasses of semirings where it does.

We start by defining epimorphisms.

Definition 110 (Homomorphism, Epimorphism)**.** *Let* $\mathcal{R}_i = (R_i, \oplus_i, \otimes_i, e_{\oplus_i}, e_{\otimes_i}), i = 1, 2$ *be two semirings. A* homomorphism *from* $\mathcal{R}_1$ *to* $\mathcal{R}_2$ *is a function* $f : R_1 \to R_2$ *s.t. for* $\odot = \oplus, \otimes$

$$f(r \odot_1 r') = f(r) \odot_2 f(r') \text{ and } f(e_{\odot_1}) = e_{\odot_2}.$$

If f *is in addition surjective, then it is an* epimorphism.

We can use them similarly to reduce problems over one semiring to problems over another semiring. Formally:

Theorem 111. *Let* $e_i(\mathcal{R}_i), i = 1, 2$ *be two encoded commutative semirings, such that*

1. *there exists a polynomial time computable epimorphism* $f : e_1(R_1) \to e_2(R_2)$*, and*
2. *for each* $e_2(r_2) \in e(R_2)$ *one can compute in polynomial time* $e_1(r_1)$ *s.t.* $f(e_1(r_1)) = e_2(r_2)$ *from* $e_2(r_2)$*.*

Then SAT$(e_2(\mathcal{R}_2))$ *is counting-reducible to SAT*$(e_1(\mathcal{R}_1))$*.*

Proof (Sketch). Given a SAT$(e_2(\mathcal{R}_2))$-instance α, we can replace all weights $e_2(r_2)$ by a weight $e_1(r_1)$ such that $f(e_1(r_1)) = e_2(r_2)$, which is possible in polynomial time by condition 2. We can evaluate the resulting SAT$(e_1(\mathcal{R}_1))$-instance and apply the epimorphism f. □

Theorem 111 formalizes the intuition which Green and Tannen [GT17] also discussed: if a semiring $\mathcal{R}_2$ is the homomorphic image of another semiring $\mathcal{R}_1$, then it is at most as information preserving as $\mathcal{R}_1$ and therefore at least as hard structurally. This is because a homomorphism f may assign different values $r \neq r' \in R_1$ the same value $f(r) = f(r') \in R_2$. Then, while we could distinguish these values in the context of $\mathcal{R}_1$, we cannot do the same in $\mathcal{R}_2$. However, clearly it cannot happen that $f(r) \neq f(r') \in R_2$ but $r = r' \in R_1$.

Note that the existence of such an epimorphism does not necessarily mean that the computational complexity of SAT$(\mathcal{R}_2)$ is lower than SAT$(\mathcal{R}_1)$ unless the other conditions of Theorem 111 are also satisfied!

Example 38. *The semiring* $\mathcal{R}_M$ *from Example 34 is a homomorphic image of* $\mathbb{B}[(x_i)_\infty]$ *using the epimorphism defined by* $f(x_i) = bin(i)$*. But SAT*$(\mathbb{B}[(x_i)_\infty])$ *is in FPSpace(exp), while, as noted before already a single multiplication in* $\mathcal{R}_M$ *may be undecidable. This does not contradict Theorem 111 since we cannot apply it as computing* $f(x_i \cdot x_j)$ *is not possible in polynomial time when* M *corresponds to the Halting problem.*

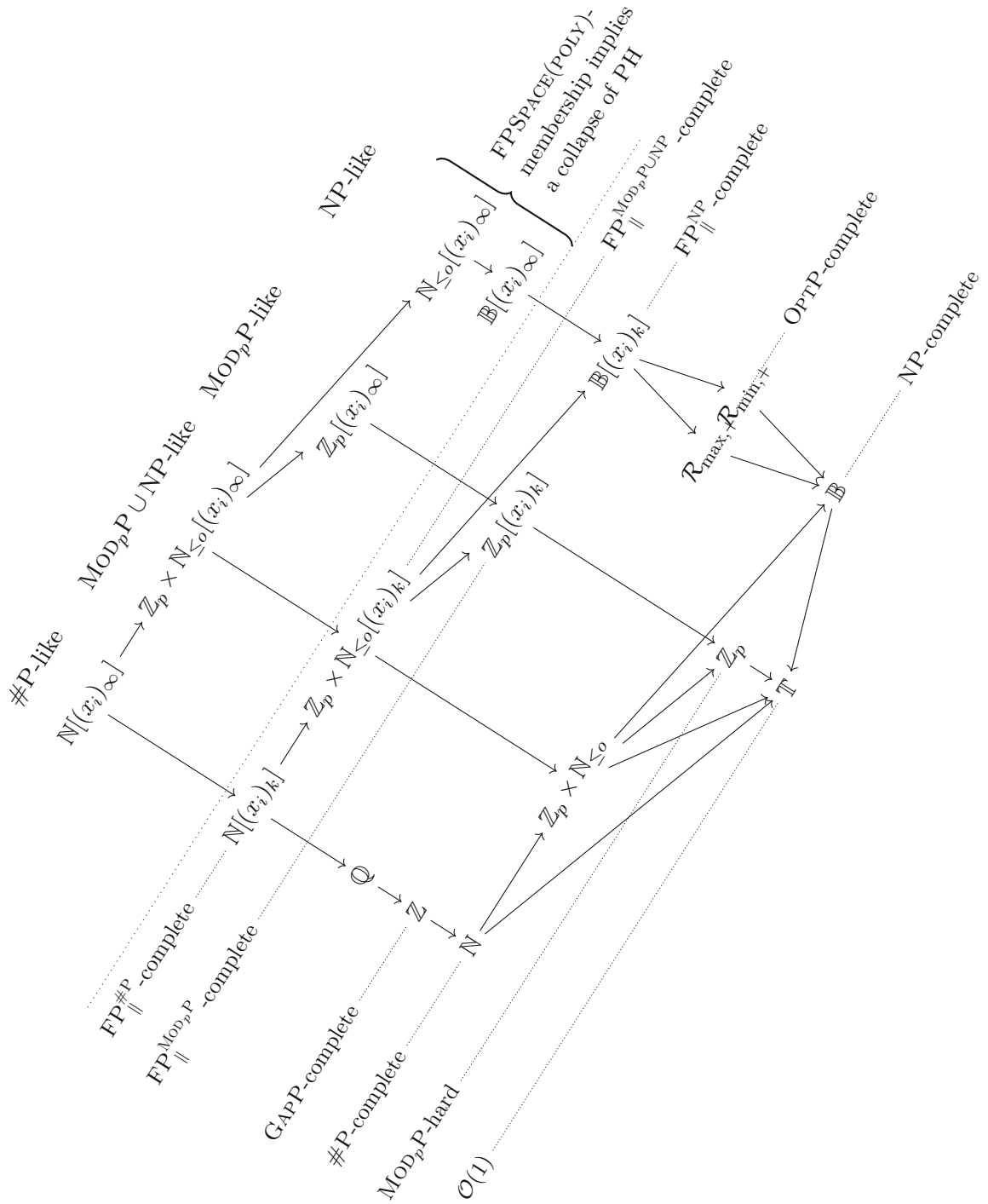


Figure 3.2: Epimorphisms $f : \mathcal{R}_1 \to \mathcal{R}_2$ between semirings, indicated by arrows $\mathcal{R}_1 \to \mathcal{R}_2$. Relation of complexity classes $\mathcal{C}$ and semirings $\mathcal{R}$, indicated by dotted lines $\mathcal{C}$ ⋯ $\mathcal{R}$.

Figure 3.2 depicts between which semirings we can hope to apply Theorem 111. There, an arrow $\mathcal{R}_1 \to \mathcal{R}_2$ indicates that there exist encoding functions e_1, e_2 such that the conditions of Theorem 111 hold. In this map, we see that $\mathbb{N}[(x_i)_\infty]$, $\mathbb{N}_{\leq o} \times \mathbb{Z}_p[(x_i)_\infty]$, $\mathbb{Z}_p[(x_i)_\infty]$ and $\mathbb{N}_{\leq o}[(x_i)_\infty]$ are the most information preserving and therefore in a sense hardest #P-like, $\text{MOD}_p\text{P} \cup$ NP-like, MOD_pP-like and NP-like commutative countable semirings, respectively. Among the four classes, we observe that the #P-like problems are the hardest and that, naturally, the $\text{MOD}_p\text{P} \cup$ NP-like problems are harder than both MOD_pP-like and NP-like problems. Furthermore, we see that by restricting ourselves to polynomials with finitely many variables, the complexity decreases (presumably) a bit for all types of problems. Finally, by further epimorphisms, we arrive at semirings that are not over polynomials, which then often correspond to some classical complexity class, as we already observed in Theorem 104.

In the following, we further formalize many intuitions, such as the $\mathcal{C}$-likeness, presented in Figure 3.2 and present further results on the complexity associated with semirings and classes thereof, often by using the existence of epimorphisms.

We start by formalizing the intuition that the above mentioned polynomial semirings are the hardest ones that satisfy a given set of properties.

Lemma 112 (Hardest Semirings)**.** *There exists an encoding e^* for*

1. $\mathbb{N}_{\leq o}[(x_i)_\infty]$;
2. $\mathbb{Z}_p[(x_i)_\infty]$;
3. $\mathbb{N}_{\leq o} \times \mathbb{Z}_p[(x_i)_\infty]$;
4. $\mathbb{N}[(x_i)_\infty]$

such that for any commutative efficiently encoded commutative semiring $e(\mathcal{R})$ that is in addition

1. *periodic with periodicity* 1;
2. *periodic with periodicity $p \geq 2$ and offset* 0;
3. *periodic with periodicity $p \geq 2$ and offset $o > 0$;*
4. *not periodic*

it holds that SAT($e(\mathcal{R})$) is counting reducible to

1. *SAT($e^*(\mathbb{N}_{\leq o}[(x_i)_\infty])$);*
2. *SAT($e^*(\mathbb{Z}_p[(x_i)_\infty])$);*

3. *SAT*$(e^*(\mathbb{N}_{\leq o} \times \mathbb{Z}_p[(x_i)_\infty]))$*;*

4. *SAT*$(e^*(\mathbb{N}[(x_i)_\infty]))$*, respectively.*

Proof (Sketch, see Appendix B.3.3 for the full proof). The idea is as follows. Instead of performing the evaluation explicitly by performing all the semiring operations on the input values immediately, we replace each semiring value $e(r)$ in the given ΣBF α by $x_{e(r)}$, where we associate the binary encoding of r with a natural number corresponding to a variable index. Then we evaluate the transformed instance over the corresponding polynomial semiring, where we obtain a polynomial as the result. Finally we use the fact that we can encode the values of the polynomial semiring in such a way that there exists a polynomial time computable epimorphism from the polynomial semiring to $e(\mathcal{R})$, which corresponds to evaluating the polynomial with $x_{e(r)}$ replaced by $e(r)$. □

Thus, if we find an efficient algorithm for $\mathrm{SAT}(e^*(\mathcal{S}[(x_i)_\infty]))$, with $\mathcal{S} \in \{\mathbb{N}_{\leq o}, \mathbb{Z}_p, \mathbb{N}_{\leq o} \times \mathbb{Z}_p, \mathbb{N}\}$ with the encoding function e^* from the proof, we can use it to solve $\mathrm{SAT}(e(\mathcal{R}))$ for any efficiently encoded commutative semiring $e(\mathcal{R})$ with the corresponding periodicity and offset. Unfortunately, already the result of evaluating a $\mathrm{SAT}(e^*(\mathbb{B}[(x_i)_\infty]))$-instance α can be exponential in the size of α. Therefore, any algorithm for $\mathrm{SAT}(e^*(\mathcal{S}[(x_i)_\infty]))$ necessarily has exponential running time.

One might be tempted to think that this can be avoided by choosing an encoding function e that is better than e^*. However, such an encoding e is unlikely to exist, as the following result shows.

Theorem 113. *Let $\mathcal{R} = \mathbb{N}[(x_i)_\infty]$ (resp. $\mathcal{R} = \mathbb{B}[(x_i)_\infty]$). If there is an encoding function e for $\mathcal{R}$ s.t.*

1) $\|[\![\alpha]\!]_{\mathcal{R}}(\emptyset)\|_e$ is polynomial in the size of α,

2) we can extract the binary representation of the coefficient $n \in \mathbb{N}$ (resp. $b \in \mathbb{B}$) of $x_{i_1}^{j_1} \ldots x_{i_m}^{j_m}$ from $e(r)$ in time polynomial in $\|r\|_e$, and

3) $\|x_i\|_e$ is polynomial in i,

then $\#P \subseteq FP/\mathrm{poly}$ (resp. $NP \subseteq P/\mathrm{poly}$).

This would imply $\Sigma_2^p = \mathrm{PH}$ [KL82], which is considered to be unlikely. Hence, for any reasonable encoding e, $\mathrm{SAT}(e(\mathbb{N}[(x_i)_\infty])$ and $\mathrm{SAT}(e(\mathbb{B}[(x_i)_\infty]))$ are unlikely to have polynomial output and are, therefore, unlikely to be in FPSpace(poly).

Condition 3) of Theorem 113 allows that indices i_k are encoded in unary and imposes no restriction on the encoding of exponents j_k. Requiring the exponents to be encoded in binary puts even already $\mathrm{SAT}(e(\mathbb{B}[x]))$ outside of FPSpace(poly), unless NP ⊆ P/poly.

Theorem 114. *Let $\mathcal{R} = \mathbb{N}[x]$ (resp. $\mathcal{R} = \mathbb{B}[x]$). If there is an encoding function e for $\mathcal{R}$ s.t.*

1) *$\|[\![\alpha]\!]_{\mathcal{R}}(\emptyset)\|_e$ is polynomial in the size of α,*
2) *we can extract the binary representation of the coefficient $n \in \mathbb{N}$ (resp. $b \in \mathbb{B}$) of x^i from $e(r)$ in time polynomial in $\|r\|_e$, and*
3) *$\|x^i\|_e$ is polynomial in $\log_2(i)$,*

then $\#P \subseteq FP/\text{poly}$ (resp. $NP \subseteq P/\text{poly}$).

Proof (Sketch, see Appendix B.3.3 for the full proofs) of Theorems 113 and 114. For each $n \in \mathbb{N}$, we can construct a ΣBF formula α of polynomial size s.t. the solution of every #3SAT (resp. 3SAT)-instance with n variables is obtainable from $\|[\![\alpha]\!]_{\mathcal{R}}(\mathcal{I})\|_e$. By the methodology of [CDS96] to assess compilability, we then infer #P $\subseteq$ FP/poly (resp. NP $\subseteq$ P/poly). □

Notably, the converse of Theorem 114 also holds:

Theorem 115. *If $\#P \subseteq FP/\text{poly}$ (resp. $NP \subseteq P/\text{poly}$), then there exist encodings e_∞ and e_1 for $\mathbb{N}[(x_i)_\infty]$ and $\mathbb{N}[x]$ (resp. $\mathbb{B}[(x_i)_\infty]$ and $\mathbb{B}[x]$) such that the preconditions 1) - 3) of Theorems 113 and 114 are satisfied.*

Proof (Sketch). The idea here is as follows: we know that we can obtain the coefficient of a monomial from $[\![\alpha]\!]_{\mathcal{R}}(\emptyset)$ in #P (resp. NP). Thus, if #P $\subseteq$ FP/poly (resp. NP $\subseteq$ P/poly), we can also obtain it in FP (resp. P) given polynomial advice.

Using this insight, we can construct an encoding that satisfies the desired properties, by representing polynomials as a tuple consisting of a SAT($e_b(\mathcal{R})$)-instance and the polynomial advice(s) necessary to solve the queries for coefficients of monomials in polynomial time. Here, we only need to make sure that the base encoding e_b satisfies condition 3). Such encodings are easy to find though. □

Thus, the existence of an encoding e as in Theorem 114 for $\mathcal{R} = \mathbb{N}[x]$ (resp. $\mathcal{R} = \mathbb{B}[x]$) is *equivalent* to the pure complexity-theoretic question whether #P $\subseteq$ FP/poly (resp. NP $\subseteq$ P/poly), which is a well-known open problem in complexity theory.

We can obtain similar results to Theorems 113 and 114 in a more general setting for polynomials with coefficients from any non-trivial commutative semiring.

Theorem 116. *Let $\mathcal{R} \neq \mathbb{T}$ be a commutative semiring. If there is an encoding function e for $\mathcal{R}[(x_i)_\infty]$ s.t.*

1) *$\|[\![\alpha]\!]_{\mathcal{R}[(x_i)_\infty]}(\emptyset)\|_e$ is polynomial in the size of α,*

2) we can extract the encoding $e(r') \in e(R)$ of the coefficient of $x_{i_1}^{j_1} ... x_{i_n}^{j_n}$ from $e(r)$ in time polynomial in $\|r\|_e$, and

3) $\|x_i\|_e$ is polynomial in i,

then either NP $\subseteq$ P/poly or $\text{Mod}_p\text{P} \subseteq$ P/poly for some $p \in \mathbb{N}$.

Theorem 117. *Let $\mathcal{R} \neq \mathbb{T}$ be a commutative semiring. If there is an encoding function e for $\mathcal{R}[x]$ s.t.*

1) $\|[\![\alpha]\!]_{\mathcal{R}[x]}(\emptyset)\|_e$ is polynomial in the size of α,

2) we can extract the encoding $e(r') \in e(R)$ of the coefficient of x^i from $e(r)$ in time polynomial in $\|r\|_e$, and

3) $\|x^i\|_e$ is polynomial in $\log_2(i)$,

then either NP $\subseteq$ P/poly or $\text{Mod}_p\text{P} \subseteq$ P/poly for some $p \in \mathbb{N}$.

As already mentioned before, if NP $\subseteq$ P/poly, then the polynomial hierarchy collapses to the second level. As a matter of fact, this also holds if $\text{Mod}_p\text{P} \subseteq$ P/poly since:

Lemma 118. *If $\text{Mod}_p\text{P} \subseteq$ P/poly for $p \in \mathbb{N}, p > 1$, then NP $\subseteq$ P/poly.*

Proof (Sketch, see Appendix B.3.3 for the full proof). We make use of the following problem:

Problem:	Unique-SAT
Input:	A propositional formula ϕ with at most one satisfying assignment.
Output:	True, if ϕ is satisfiable, otherwise false.

Note that this is a *promise problem*: the input is guaranteed to have at most one satisfying assignment, if this is not the case any output, including non-termination are allowed. It follows, that for any $p \in \mathbb{N}, p > 1$ we can solve Unique-SAT in Mod_pP by asking whether the number of satisfying assignments of the input is not equivalent to zero modulo p.

Valiant and Vazirani [VV86] showed that $\text{NP} \subseteq \text{RP}^{\text{Unique-SAT}}$. Using RP $\subseteq$ BPP we get $\text{RP}^{\text{Unique-SAT}} \subseteq \text{BPP}^{\text{Unique-SAT}}$ and from Unique-SAT $\in \text{Mod}_p\text{P}$ for any $p \in \mathbb{N}, p > 1$ it follows that $\text{NP} \subseteq \text{BPP}^{\text{Mod}_p\text{P}}$.

It remains to show that $\text{BPP}^{\text{Mod}_p\text{P}} \subseteq$ P/poly follows from $\text{Mod}_p\text{P} \subseteq$ P/poly. We already know that BPP $\subseteq$ P/poly [BG81], even without the assumption that $\text{Mod}_p\text{P} \subseteq$ P/poly. We establish that $\text{P}^{\text{P/poly}}/\text{poly} \subseteq$ P/poly and $\text{BPP}^{\text{Mod}_p\text{P}} \subseteq \text{P}^{\text{P/poly}}/\text{poly} \subseteq$ P/poly, which proves the result. □

It follows that if there is any non-trivial commutative semiring $\mathcal{R}$ with an encoding e as required by Theorems 116 or 117 then NP $\subseteq$ P/poly. This would imply a collapse of the polynomial hierarchy and is therefore considered to be unlikely. The full proofs of Theorems 116 and 117 are given in Appendix B.3.3.

Summing up, we see that reducing SAT($e(\mathcal{R})$) to the polynomial semirings is not practical for encoded semirings $e(\mathcal{R})$ in general[4]. However, we can obtain positive results when restricting ourselves to polynomials with a limited number of variables by using a different encoding.

Theorem 119. *Let $e(\mathcal{R})$ be a commutative semiring that is efficiently encoded. Then SAT($e(\mathcal{R}[(x_i)_k])$) is $\mathrm{FP}_{\|}^{\mathrm{NP}(e(\mathcal{R}))}$-complete for metric reductions, if we extend e to $\mathcal{R}[(x_i)_k]$ by representing polynomials as lists of monomials with exponents in unary and coefficients encoded by e.*

As mentioned before the semirings of polynomials with coefficients in $e(\mathcal{R})$ allow one to solve multiple SAT($e(\mathcal{R})$)-instances at the same time. Theorem 119 shows that when we choose a specific encoding and allow at most k variables then the number of instances that can be solved using one instance over the corresponding semiring $e(\mathcal{R}[(x_i)_k])$ is exactly polynomial.

Proof (Sketch, see Appendix B.3.3 for the full proof). We can obtain the coefficient of the polynomially many monomials in the solution of an SAT($e(\mathcal{R}[(x_i)_k])$)-instance with one independent query to an NP($e(\mathcal{R})$)-oracle each. On the other hand we can solve n SAT($e(\mathcal{R})$)-instances at the same time by solving one SAT($e(\mathcal{R}[(x_i)_k])$)-instance, by multiplying each of them with a different monomial x_1^i. □

As an immediate consequence, we see that SAT(bin($\mathbb{Q}[(x_i)_k]$)) and SAT(bin($\mathbb{B}[(x_i)_k]$)) are metrically reducible to $\mathrm{FP}_{\|}^{\#\mathrm{P}}$ (which is equal to $\mathrm{FP}^{\#\mathrm{P}[1]}$) and $\mathrm{FP}_{\|}^{\mathrm{NP}}$, respectively, and thus not significantly harder than #P and NP, respectively.

The same can be said when a SAT($e(\mathcal{R})$)-instance only contains few values from the semiring.

Lemma 120. *Let $k \in \mathbb{N}$ be fixed, $e(\mathcal{R})$ be an efficiently encoded commutative semiring. If $e(\mathcal{R})$ is*

1. *periodic with periodicity 1;*
2. *periodic with periodicity $p \geq 2$ and offset 0;*
3. *periodic with periodicity $p \geq 2$ and offset $o > 0$;*

[4]Note that Knowledge Compilation [DM02] intuitively does something very similar but optimizes the size of the representation of the polynomials such that it can be bounded using the treewidth of a given formula.

4. *not periodic*

then SAT($e(\mathcal{R})$) for instances that contain at most k different semiring values is in

1. $FP_{\|}^{NP}$;
2. $FP_{\|}^{\text{Mod}_p P}$;
3. $FP_{\|}^{\text{Mod}_p P \cup NP}$;
4. $FP_{\|}^{\#P}$

respectively.

Proof. Let α be a SAT($e(\mathcal{R})$)-instance that contains at most k different semiring values $e(r_1), \ldots, e(r_k)$. We define an epimorphism $f : \text{bin}(\mathbb{N}[(x_i)_k]) \to e(\mathcal{R})$ by defining it on the monomials $x_1^{i_1} \ldots x_k^{i_k}$ as $f(x_1^{i_1} \ldots x_k^{i_k}) = \bigotimes_{j=1}^{k} \bigotimes_{l_j=1}^{i_j} e(r_j)$. Since $e(\mathcal{R})$ is efficiently encoded, and the exponents i_j are encoded in unary, f is computable in polynomial time. We can thus apply Theorem 111 with semirings $\text{bin}(\mathbb{N}[(x_i)_k])$ and $e(\mathcal{R})$, since the second condition is trivially satisfied by using x_j as an element that maps to $e(r_j)$ for $j = 1, \ldots, k$.

Thus, we have a counting reduction from SAT($e(\mathcal{R})$) to SAT($\text{bin}(\mathbb{N}[(x_i)_k])$). By applying Theorem 119 we then obtain a metric reduction to a problem in $\text{FP}_{\|}^{\#\text{P}}$. Since metric reductions are polynomial time functions, we can derive that SAT($e(\mathcal{R})$) is in $\text{FP}^{\text{FP}_{\|}^{\#\text{P}}[1]}$, which is equal to $\text{FP}_{\|}^{\#\text{P}}$.

The proof for the semirings that are periodic is analogous by performing the same reduction to

1. SAT($\text{bin}(\mathbb{N}_{\leq o}[(x_i)_k])$), when $e(\mathcal{R})$ has periodicity 1 and offset o;
2. SAT($\text{bin}(\mathbb{Z}_p[(x_i)_k])$), when $e(\mathcal{R})$ has periodicity $p \geq 2$ and offset 0;
3. SAT($\text{bin}(\mathbb{Z}_p \times \mathbb{N}_{\leq o}[(x_i)_k])$), when $e(\mathcal{R})$ has periodicity $p \geq 2$ and offset $o > 0$.

□

However, not all relevant SAT($\mathcal{R}$)-instances have few different semiring values. We can take this idea even further by considering instances that have many different semiring values that can be expressed as "reasonably small" sums and products of few different semiring values. For this, we consider finitely generated semirings.

Definition 121 (Finitely Generated Semiring)**.** *Let $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ be a semiring. We call $\mathcal{R}$* finitely generated*, if $\mathcal{R} = \langle R^\star \rangle_\mathcal{R}$ for some finite $R^\star \subseteq R$.*

Commutative semirings that are finitely generated using k elements $r_1, \dots r_k$ can be seen as reduced versions of the semiring of polynomials with variables $x_1, \dots, x_k$. We thus obtain:

Theorem 122. *Let $e(\mathcal{R})$ be an efficiently encoded commutative semiring that is generated by $\{r_1, \dots, r_k\}$. Suppose every $r \in R$ is of the form $r = \bigoplus_{i=1}^{n} a_i \cdot \bigotimes_{j=1}^{k} r_j^{e_{i,j}}$ for some $a_i, e_{i,j} \in \mathbb{N}$ such that*

- *$\max\{e_{i,j}, \log_2(a_i), n\}$ is polynomial in $\|r\|_e$, and*
- *we can obtain $a_i, e_{i,j}$ from $e(r)$ in polynomial time.*

If $e(\mathcal{R})$ is

1. *periodic with periodicity 1;*
2. *periodic with periodicity $p \geq 2$ and offset 0;*
3. *periodic with periodicity $p \geq 2$ and offset $o > 0$;*
4. *not periodic;*

then SAT($e(\mathcal{R})$) is in

1. $FP_{\|}^{NP}$;
2. $FP_{\|}^{\text{Mod}_p P}$;
3. $FP_{\|}^{\text{Mod}_p P \cup NP}$;
4. $FP_{\|}^{\#P}$;

respectively.

Note that k in Theorem 122 may be zero. For example, the natural number semiring $\mathbb{N}$ is generated by the empty set, since every semiring over a subset over the natural numbers with the same addition and multiplication needs to contain zero and one. Clearly, $\mathbb{N} = \langle 0, 1 \rangle_\mathbb{N}$. If indeed $k = 0$, the product $\bigotimes_{j=1}^{k} r_j^{e_{i,j}}$ takes values $e_\otimes$ and the Theorem also holds.

Proof (Sketch, see Appendix B.3.3 for the full proof). As in the proof of Lemma 120, we can find a polynomial time epimorphism from $e(\mathcal{R})$ to $\text{bin}(\mathcal{S}[(x_i)_k])$ for the semiring $\mathcal{S} \in \{\mathbb{N}_{\leq o}, \mathbb{Z}_p, \mathbb{Z}_p \times \mathbb{N}_{\leq o}, \mathbb{N}\}$ with the same periodicity as $e(\mathcal{R})$. However, contrary to Lemma 120, we do not identify each semiring value in a given instance with a different variable but identify the values $r_1, \ldots, r_k$ that the semiring is generated from with the variables $x_1, \ldots, x_k$. Due to the preconditions of the Theorem, we can then apply Theorem 111 to show that we can counting-reduce $\text{SAT}(e(\mathcal{R}))$ to $\text{SAT}(\text{bin}(\mathcal{S}[(x_i)_k]))$. By application of Theorem 119 we then prove our claim.

Again note that in case $k = 0$, the polynomial semiring $\text{bin}(\mathcal{S}[(x_i)_k])$ is simply the semiring over the constant polynomials, i.e., $\text{bin}(\mathcal{S})$. □

Note that when $e(\mathcal{R})$ is periodic, we know that the coefficients a_i can always be chosen such that they are bounded by a constant.

Furthermore, compare Theorem 122 to Theorem 109 for the case when the encoded semiring $e(\mathcal{R})$ is not periodic. Clearly, for any semiring $e(\langle e_\otimes \rangle)$ is finitely generated. Thus, one might be tempted to assume that one can always apply Theorem 122 to derive that $\text{SAT}(e(\langle e_\otimes \rangle))$ is in $\text{FP}_{\|}^{\#\text{P}}$. This is not necessarily the case, however, since it is not clear that the additional preconditions of Theorem 122 are given, even when the semiring is efficiently encoded.

3.7 Related Works

In order to capture the complexity of quantitative reasoning frameworks that are defined over semirings $\mathcal{R}$, we introduced Semiring Turing Machines. SRTMs are non-deterministic Turing machines whose result is determined by their transitions that are weighted with semiring values; $\text{SAT}(\mathcal{R})$ and $\text{NP}(\mathcal{R})$, the versions of SAT and NP that are generalized to semirings are in the expected relation, viz. that $\text{SAT}(\mathcal{R})$ is $\text{NP}(\mathcal{R})$-complete with respect to (adjusted) Karp-reductions.

In the literature, other extensions of machine models with means to deal with algebraic structures have been considered. In this section, we consider these related models of computation with algebraic structures.

Weighted Automata The machine model most related to our SRTMs is that of weighted automata, which are finite state automata that have a (semiring)-weighted transition function [DG07]. SRTMs are a generalization of weighted automata and can simulate the latter with ease; informally, they equip weighted automata with memory and the full computational power of a Turing machine under the restriction that semiring values are opaque. The connection to weighted automata is especially of interest due to their connection to descriptive complexity, which opens up future work on the same topic for SRTMs.

Blum-Shub-Smale Machines Next, our SRTMs are related to Blum-Shub-Smale (BSS) machines, which were introduced by Blum, Shub, and Smale; Blum [BSS89; Blu98] to model algebraic computation over the reals and general rings, respectively. Roughly speaking, BSS machines operate over a tape whose cells can hold elements from a ring R, and computation steps are performed by the evaluation of polynomials over R that modify finite sequences of cells; a BSS machine is a connected directed graph which has input, output, and computation nodes, and possibly shift and branching nodes, where the former allow for shifting tape contents and the latter for testing a condition such as $r \geq 0$ (if the ring R is ordered) or $r = 0$ based on the ring elements on the tape. Notably, Blum et al. remarked that a classical Turing machine is a BSS-machine over $\mathbb{Z}_2$ (using a binary alphabet).

As for complexity, BSS studied in their model ring operations having unit cost, i.e., independent of the concrete ring elements used; merely for $\mathbb{Z}$ and $\mathbb{Q}$, also bit cost, i.e. $\log m$ where m is the largest number in the computation, was considered. Based on this time measure, Blum et al. defined then polynomial (P-) time and NP computations over rings, where in the latter a guess of values from R can be made; notably, over $\mathbb{Z}$ we have $\mathrm{P} \neq \mathrm{NP}$ under unit cost.

Compared to SRTMs, BSS-machines do not operate over semirings, thus e.g. the natural numbers $\mathbb{N}$ are not covered, as they do not form a ring. While the definition might be adapted to semirings, a distinctive feature is that BSS-machines work over (strings of) ring values only, while our SRTMs allow for semiring values in addition to a tape alphabet as usual. The latter is convenient and provides flexibility for problem representation, which the BSS model lacks. E.g., for solving the classical SAT problem in the BSS model, propositional formulas must be first converted into an algebraic form over a ring that can be processed by a BSS machine; to this end, Blum et al. use a transformation (assuming that input formulas are already in CNF) into a system of polynomials over the ring $\mathbb{Z}_2$.

Furthermore, in order to ensure efficient (polynomial time) computation, the processing of numbers must be controlled with additional restrictions, such that e.g. recursive multiplication of values, as discussed in Section 3.4.2, is prohibited.

Summarizing, the BSS model has been devised with computations over algebraic structures in mind, where the input is already in a specific form and the cost of algebraic operations is disregarded respectively for $\mathbb{Z}$ and $\mathbb{Q}$ estimated by binary number size; semirings like $\mathbb{N}$ are not covered. Efficient computation on classical Turing machines is hard to ensure, and the linkage to conventional complexity classes is limited to few cases, and different by missing or fixed encodings. This makes the use of BSS machines for problem solving and complexity analysis in the context of AI reasoning frameworks as we consider less attractive.

Real Counting Computations Major motivations of Blum et al. to develop their machine model were to have an idealized computation model for scientific computation with real numbers, in which the cost of multiplication is independent of the size of

numbers (which on a Turing machine depends on the encoding), and to bring the theory of computation into the domain of analysis, geometry and topology, such that the mathematics of these subjects can then be put to use in the systematic analysis of algorithms [BSS89]. Building on the BSS machine model, Meer [Mee00] developed a theory of counting problems over the real numbers. To this end, he introduced the class $\#\mathrm{P}_{\mathbb{R}}$ as the real analogue of #P, which also counts accepting paths but of a non-deterministic BSS machine, and then embarked on giving a logical characterization of $\#\mathrm{P}_{\mathbb{R}}$ following descriptive complexity theory over the reals, which has been initiated in [GM95]. Meer [Mee00]'s class $\#\mathrm{P}_{\mathbb{R}}$ is different from our $\mathrm{NP}(\mathcal{R})$ instantiated for $\mathcal{R} = \mathbb{R}$, in which we sum up (i.e., "count") *weighted* acceptance paths. However, for SRTMs the numbers involved in a computation are deliberately restricted compared to BSS machines, in order not to produce large result values (which does not matter in the BSS-model) such that space efficient storage on a conventional machine is feasible. Detailing the relationship between $\mathrm{NP}(\mathbb{R})$ and an analogue class that combines weighted acceptance counting with BSS machines is an interesting issue for future research, especially when it comes to generalizations for other (semi)rings.

Real Counting with Arbitrarily Approximatable Weights A different definition of $\#\mathrm{P}_{\mathbb{R}}$ and $\#\mathrm{P}_S$ for $S \subseteq \mathbb{R}$ was provided by De Campos, Stamoulis, and Weyland (2020). While they intuitively also define the output of a computation as the sum over weighted computation paths like we do, they do not assume the values to be given explicitly but require functions that compute them up to a variable precision of 2^{-p} in polynomial time in p. This allows them to work with a finite representation of real numbers. Interestingly, this different way of encoding numbers leads to undecidability of the question whether a computation in $\#\mathrm{P}_{\mathbb{Q}}$ leads to a value greater than zero. Therefore, we argue that while this matter of encoding numbers is interesting to allow computations with uncountable semirings such as $\mathbb{R}$ on classical Turing machines, the explicit encoding in our setting is of more interest when the semiring has only countably many values, as it leads to more intuitive results.

Relational Machines SRTMs are further related to the relational machine of Abiteboul and Vianu [AV91], which was introduced in order to compensate for a mismatch between computations by a Turing machine and the evaluation of logical formulas over finite relational structures, which is of central importance for query answering over relational databases. The generic treatment of elements in such structures, which model relational databases, by logic and the fact that relations are unordered sets of tuples is in contrast to string-based representations where in an encoding elements might be distinguished and by the inherent order of the string's letter an order of tuples induced; the latter may make it easy to answer certain queries (e.g., whether a database has an even number of records) that can not be expressed in a query language without order information (e.g., pure SQL). To this end, Abiteboul and Vianu [AV91] extended Turing machines with a "relational storage" and the ability to perform relational operations (which underlie the evaluation of logical formulas) directly on the relational storage rather than on the

encoding of a database on a machine's tape. Emerging relational complexity classes and their relationships allowed for a precise characterization of the expressive powers of certain database query languages and to translate relationships of the latter into equivalent questions in standard complexity theory. Notably, our notion of Karp surrogate-reduction for problems over semirings aims at a genericity condition similar to the one for databases, but it does not allow for exploiting in the reduction that different semiring surrogates SV^nE and SV^mE (i.e., $n \neq m$), which corresponds to having different elements in a database, will amount to different semiring values.

3.8 Discussion

Just like the quantitative frameworks that are defined over semirings $\mathcal{R}$ capture many different quantitative problems when instantiated with the correct semiring, we showed that the same is the case on the complexity level. Theorem 104 showed that for $(\mathcal{R}, \mathcal{C}) = (\mathbb{B}, \mathrm{NP})$, $(\mathbb{N}, \#\mathrm{P})$, $(\mathbb{Z}, \mathrm{GAPP})$, $(\mathcal{R}_{\max,+}, \mathrm{OPTP})$ and the binary representation bin of the integers, $\mathrm{SAT}(\mathrm{bin}(\mathcal{R}))$ is $\mathcal{C}$-complete w.r.t. Karp reductions. Thus, many well-known complexity classes can be seen as instantiations of $\mathrm{NP}(\mathcal{R})$ with the appropriate semiring under the usual representation of numbers.

Using $\mathrm{NP}(\mathcal{R})$, we are able to characterize the complexity of many semiring frameworks. As our results show that

- Algebraic Measures Section 2.2,
- Sum-Of-Products Problems [BDP09],
- Weighted First-Order Logic Evaluation [MR15], Section 2.3,
- Semiring-based Constrained Satisfaction Problems [Bis+99],
- Algebraic Model Counting [KVD17], and
- Semiring-induced Propositional Logic [LOR10]

are all $\mathrm{NP}(\mathcal{R})$-complete with respect to Karp-reductions. Furthermore, we saw that Semiring Provenance [GKT07] for Datalog is $\mathrm{NP}(\mathcal{R})$-hard but not complete and therefore even harder than other semiring formalisms.

Applying our Results for Problem Analysis As showcases, we consider some specific semirings to illustrate how our previous results can be used to analyze the complexity of further specific semirings.

As a consequence of Theorems 111 and 119, or directly from Theorem 122 for the finitely generated $\mathbb{N}, \mathbb{Z}$, we obtain:

Theorem 123. *Let $\mathbb{S} = \mathbb{N}, \mathbb{Z}, \mathbb{Q}$. For $\mathbb{S}^n$, $n \in \mathbb{N}$, the semiring $\mathbb{S}$ over multiple dimensions, we have that SAT(bin$(\mathbb{S})^n$) is $\mathrm{FP}_{\|}^{\#\mathrm{P}}$-complete with respect to metric reductions, where bin$(\mathbb{Q})$ represents $r \in \mathbb{Q}$ as pair $(\mathit{bin}(p), \mathit{bin}(q))$ such that $p/q = r$, $p \in \mathbb{Z}$, $q \in \mathbb{N}$ and the greatest common divisor of $|p|$ and q is 1.*

As promised in the introduction, we consider in more detail the semiring

$$\mathrm{GRAD} = (\mathbb{Q}_{\geq 0} \times \mathbb{Q}, +, \otimes, (0,0), (1,0)),$$

where addition is coordinate-wise and $(a_1, b_1) \otimes (a_2, b_2) = (a_1 \cdot a_2, a_2 \cdot b_1 + a_1 \cdot b_2)$. It was introduced by [Eis02] and shown to be useful for parameter optimization [KVD17; Man+21]. Intuitively, we can compute gradient values with GRAD and for (a, b) the value a corresponds to the computed result of some function and b corresponds to its gradient with respect to some parameter.

There is an epimorphism f from $\mathbb{Q}[x]$ to GRAD, which can be seen by defining f on the monomials qx^i as

$$f(qx^i) = \left\{ \begin{array}{cc} q(0,1)^i & \text{if } i > 0, \\ |q| & \text{otherwise.} \end{array} \right\} = \left\{ \begin{array}{cc} (0,q) & \text{if } i = 1, \\ (0,0) & \text{if } i > 1, \\ (|q|,0) & \text{otherwise.} \end{array} \right.$$

This means that we can see the elements in GRAD as elements in $\mathbb{Q}[x]$ by identifying $(1,0)$ and $(0,1)$ with 1 and x, respectively. The elements in GRAD are however only reduced versions of the polynomials, i.e., there are additional equalities between values in GRAD that do not hold in $\mathbb{Q}[x]$. An example is x^2, because $(0,1) \otimes (0,1) = (0,0)$ but $x^2 \neq 0$.

Theorem 124. *SAT(GRAD) is $\mathrm{FP}_{\|}^{\#\mathrm{P}}$-complete with respect to metric reductions.*

Here, the membership follows from Theorems 119 and 122 while the hardness follows from the $\mathrm{FP}_{\|}^{\#\mathrm{P}}$-hardness of bin$(\mathbb{N})$ and the fact that $f : \mathrm{GRAD}_{\mathbb{N}} \to \mathbb{N}, (a,b) \mapsto a$, where $\mathrm{GRAD}_{\mathbb{N}}$ is the semiring GRAD restricted to elements from $\mathbb{N} \times \mathbb{N}$, is an epimorphism. The latter implies that we can use Theorem 111 to prove the hardness part. The full proof can be found in Appendix B.3.4.

Effects of the Encoding By considering the complexity of $\mathrm{SAT}(e(\mathcal{R}))$ for encoded semirings $e(\mathcal{R})$ in terms of classical complexity classes, we are able to gain a broad overview of how the complexity depends on the semiring. The main intuition that we gain is that the more information semiring values can preserve in addition and multiplication the harder it is to evaluate $\mathrm{SAT}(e(\mathcal{R}))$, which we formalized using epimorphisms in Theorem 111. In the simplest case, where we consider $\mathrm{SAT}(e(\langle e_\otimes \rangle))$, instances only have weights of the form $e(k \cdot e_\otimes)$ and can thus only retain very limited information. Here, there are exactly four different types of non-trivial problems that can occur. Namely, NP-, $\mathrm{Mod}_p\mathrm{P}$-, $\mathrm{Mod}_p\mathrm{P} \cup \mathrm{NP}$- or #P-like problems, which are not only similarly hard

as the corresponding complexity class but can also be solved (under some additional restrictions on the encoding function e) with a corresponding oracle.

Interestingly, the separation by problem type is already determined by the periodicity and offset of the semiring and also applies to the case where general weights are allowed, if the semiring is efficiently encoded. Namely, there exists an encoding e^* such that all NP-, Mod_pP-, $\text{Mod}_p\text{P} \cup \text{NP}$- and #P-like problems can, respectively, be counting reduced to $\text{SAT}(e^*(\mathcal{R}[(x_i)_\infty]))$ for $\mathcal{R} \in \{\mathbb{N}_{\leq o}, \mathbb{Z}_p, \mathbb{N}_{\leq o} \times \mathbb{Z}_p, \mathbb{N}\}$. This means that these semirings are in a sense the hardest ones in this class of problems. In terms of information, the values of these polynomial semirings intuitively carry the results of multiple $\mathcal{C}$-queries, by handling one $\mathcal{C}$-query per monomial, where $\mathcal{C}$ is the corresponding complexity class of the problem type.

Unfortunately, this is likely too much information to store in a "reasonable" polynomial size output. For each $n \in \mathbb{N}$ we are able to construct a $\text{SAT}(e(\langle e_\otimes \rangle_{\mathcal{R}[(x_i)_\infty]}))$-instance of size polynomial in n, whose result contains as information the solution of every $\text{SAT}(e(\langle e_\otimes \rangle_{\mathcal{R}}))$-instance without weights of size at most n. Since there are exponentially many such instances, this would mean we would need to compute in some way the solution of exponentially many $\mathcal{C}$-instances. Using this insight we showed that the existence of a "reasonable" encoding e such that $\text{SAT}(e(\mathcal{R}[(x_i)_\infty]))$ is in $\text{FPSpace}(\text{poly})$ would imply a collapse of the polynomial hierarchy and is, thus, considered to be unlikely.

While this strategy of reducing all $\mathcal{C}$-like problems to the corresponding polynomial semiring $e(\mathcal{R}[(x_i)_\infty])$ does not seem recommendable, we see that with appropriate encodings e the semiring $e(\mathcal{R}[(x_i)_k])$ of the polynomials with at most k variables only allows us to retrieve information about polynomially many calls to a $\mathcal{C}$-oracle. We can exploit this result to show that all $\mathcal{C}$-like problems with few weights from the semiring as well as all $\mathcal{C}$-like problems over a finitely generated semiring with a suitable encoding can be solved in $\text{FP}_{\|}^{\mathcal{C}}$.

Summing up, our results can be interpreted to give the following intuition: Putting aside the difficulties caused by the encoding e, the non-deterministic evaluation problems over non-trivial semirings $\mathcal{R}$ are generally hard, as they are strongly intertwined with one of NP, Mod_pP, $\text{Mod}_p\text{P} \cup \text{NP}$ and #P based on the periodicity and the offset of $\mathcal{R}$. Within the class of $\mathcal{C}$-like problems P the number of queries to an $\mathcal{C}$-oracle, whose results can be obtained from the solution of P, in relation to the size of the P-instance, seems to be crucial. When it is polynomial an $\text{FP}_{\|}^{\mathcal{C}}$-computation is sufficient, when it is exponential, an $\text{FPSpace}(\text{exp})$-computation seems to be necessary.

3.9 Conclusion

In this chapter, we have presented a model of computation that equips Turing machines with weighted transitions using values from a semiring, and we have defined with $\text{NP}(\mathcal{R})$ an analogue of NP over generic semirings $\mathcal{R}$; specific instances can be used to characterize the complexity of a number of problems in quantitative reasoning frameworks

in AI. Furthermore, we have linked the novel semiring complexity classes to standard complexity classes, by looking into the encoding of a semiring for conventional, string-based computing.

Our investigation of semiring complexity motivated by AI problems is by no means exhaustive. On the contrary, there are a number of further research issues that emerge from it that remain for future work.

Other Classes of Semirings In this work, we have focused on commutative semirings and restrictions on the addition of the semiring. Dropping commutativity should lead to problems that are at least as hard as in presence of commutativity according to Theorem 111, since we can find epimorphisms from non-commutative to commutative semirings but in general not the other way around. While some of our results do not make use of commutativity, others such as Theorem 122 rely on it. Furthermore, periodicity and offset are properties of the addition of the semiring. If we impose additional conditions on multiplication, we may be able to exploit them for better upper bounds on the complexity of the evaluation problem.

BSS Machines and SRTMs BBS machines and SRTMs both provide a way to perform computations over algebraic structures but SRTMs are more restricted. A characterization of when the machine models differ in power depending on which additional restrictions are put on the BSS machines will help us better understand the effect of limitations on the computational model. Especially, a closer look at the difference in power attained by allowing SRTMs to perform recursive computations with semiring values on the tape is of interest. While we have seen that over natural numbers $\mathbb{N}$ this leads to strictly more powerful machines –as we can compute a value that is double exponential in the size of the input– this is not clear for other semirings in general. For example, over the Boolean semiring $\mathbb{B}$ recursive reuse of semiring values in computations does not increase the hardness since CircuitSAT is Karp-reducible to SAT and vice versa. This begs the question of which properties a semiring has to satisfy such that recursive semiring computations are harder than non-recursive ones.

Semiring Counting Hierarchy Another issue of computational models over algebraic structures that would deserve investigation is an analogue of the polynomial hierarchy for general semirings, going beyond the one for counting problems over the natural numbers as in the work of Allender and Wagner [AW93]. For example, the problem MajMajSAT requires the solution of a counting problem over the natural numbers $\mathbb{N}$ on the second level, cf. [OCD16a]. Further problems such as computing the maximum expected utility of a logical theory, probabilistic reasoning over stable models, Maximum A Posteriori (MAP) inference and some decision theoretic problems are also intrinsically second-level problems [KTK22] but use different semirings at the different levels. The FAQ problem [KNR16] lifts this to problems at an arbitrary level, by allowing for sequences of semiring aggregates. It would thus be interesting to enhance the power of SRTMs such that we can solve and characterize the complexity of such multi-level semiring reasoning frameworks.

A promising starting point for this direction of research is the approach of Ladner [Lad89] who introduced a counting version of the polynomial hierarchy based on alternating Turing Machines.

Descriptive Complexity Apart from the computational complexity of logics over semirings, there is also the question of descriptive complexity. Here, Droste and Gastin [DG07] showed that a fragment of weighted monadic second order logic was shown to capture the sentences recognizable by weighted automata over semirings. This result is a generalization of the Büchi, Elgot-Trakhtenbrot Theorem [Büc60], which states the same in the unweighted (i.e. Boolean) case. This suggests that we may be able to do the same for Fagin's seminal theorem [Fag74] in descriptive complexity, which states that the existential fragment of second-order logic characterizes the complexity class NP. Recently, second-order logics over specific semirings were already used to capture the complexity of quantitative complexity classes such as FP and #P [AMR20]. A general theorem that relates NP($\mathcal{R}$) and second-order logic over semirings $\mathcal{R}$ would be suggestive. Given that SRTMs generalize weighted automata with their semiring weighted transition functions, our work provides a basis for a generalized version of Fagin's Theorem over arbitrary semirings. Such a result and others on descriptive complexity would be valuable to assess the expressiveness of AI reasoning frameworks over varying input data when formulas respectively queries are fixed.

Polynomial Semirings for Knowledge Compilation Another interesting avenue of research is the application of polynomial semirings in the context of Knowledge Compilation. Intuitively, many contemporary approaches in that area compile a circuit that represents a value from $\mathbb{B}[(x_i)_\infty]$ as succinctly as possible without losing information, cf. [DM02; KVD17; DPV20]. In an application setting such as probabilistic reasoning or most probable explanation inference, this however may be way more information than necessary. A way to mitigate this problem may be given by semirings $\mathcal{R}[(x_i)_k]$ with finitely many variables, which our results suggest may allow for representations that are lot smaller. This would not only be of theoretical benefit but may even lead to more efficient inference in practical applications.

CHAPTER 4

Efficient Algebraic Answer Set Counting

In this chapter, we consider the task of efficiently performing quantitative reasoning over the set of models in ASP.

In the theoretical analysis of the computational complexity of counting problems over semirings, we have seen that the most rewarding strategy to tackling *Algebraic Answer Set Counting (AASC)*, i.e., the evaluation of queries with algebraic measures, over general semirings is likely to adapt the used tools to the specific semiring. Thus, we are faced with a dilemma: on the one hand, we aim to provide an implementation that works for general semirings, on the other hand, we want to improve the performance of reasoning. Doing both to the fullest extent would imply finding more efficient strategies for each of the different settings that we identified above. We consider this infeasible within the scope of this thesis.

We compromise and study strategies applicable to general semirings in the hope to provide improved performance when AASC over the given semiring is #P-hard. The latter can be achieved in an implementation for AASC over general semirings, since current strategies for the #P-hard setting naturally generalize to the general setting [KVD17].

Notably, the #P-hard setting includes many relevant and recent frameworks for probabilistic reasoning, including LP^{MLN} [LY17], P-log [BGR09], ProbLog [DKT07], PITA [RS11], CP-logic [VDB09], SMProbLog [TKR23] but also many others [NM14; RC22; dMF13]. Going beyond probabilistic inference for a fixed distribution features *Neuro-Symbolic Reasoning*, for example in NeurASP [YIL20], SLASH [Skr+22], or DeepProbLog [Man+21].

Since there already exists a wide range of implementations and diverse approaches to AASC and especially probabilistic inference [Fie+15; Fie+11; RS11; LTW17; Hah+22; Fic+17; Vla+16; Vla+14] it makes sense to improve upon an existing idea. However,

there is no consensus regarding which evaluation strategy is generally favorable.Thus, it is initially unclear which approach to choose as the basis of our work.

In *enumeration*-based approaches [YIL20; Hah+22; LTW17] one enumerates all the answer sets one after the other and sums up their weights one by one. While the idea is simple, it works well for "few" (less than a billion) answer sets. However, it does not scale when there are many answer sets, even for structurally very simple programs. *Knowledge Compilation*-based [DM02; Men21] approaches [Fie+11; Vla+14; RS11] are more sophisticated and produce a so called *tractable circuit representation* of a given program. They represent the answer sets of a program in a manner that can be vastly more succinct than an explicit representation of all the answer sets but still allow us to perform AASC using linearly many semiring operations in the size of the resulting circuit [KVD17]. Apart from that, there is also the option to perform dynamic programming [Fic+17], when the *treewidth* of the program is low enough. Here, treewidth is a parameter that intuitively gives an upper-bound on how complex the structure of a program (or more generally a graph) is by measuring how "treelike" the shape of rule dependencies is, which determines how expensive it is to decompose the problem into smaller ones in a recursive manner.

However, there are not only different basic approaches, but for example for Knowledge Compilation there are different classes of tractable circuits to compile to, we can compile some circuits either in a "bottom up" [Dar11; Som12] or in a "top down" [OD15; Dar04; KJ21] manner, and we can either compile the program directly or convert it into a propositional formula in Conjunctive Normal Form (CNF) first.

In a first step, we therefore investigate different strategies in depth from a theoretical point of view using recent results from the field of parameterized complexity [OD14; BS17; FPV05; JS12; Ama+20]. While these results do not give us general lower bounds they do give us upper bounds in terms of structural parameters, such as treewidth or pathwidth, which we can use to derive performance guarantees for the considered strategies. Based on this analysis, we conclude that the most promising strategy is to first translate programs into CNFs and compile them using one of the highly advanced tools from the SAT community [Dar04; OD15; LM17; KJ21].

Naturally, this CNF translation should be chosen in such a manner that it leads to good Knowledge Compilation performance. From the previous theoretical analysis, we know that small CNFs of low treewidth allow for good worst case guarantees. In fact, there is even recent empirical evidence that shows that explicitly exploiting the treewidth of CNFs during Knowledge Compilation can lead to significantly improved performance [KJ21]. Apart from the size and treewidth, we assume that it is beneficial if the CNF encoding is not very "semantically complex" such that intuitively the knowledge compiler can easily identify parts of the search space that it may discard. This last measure is rather imprecise, nevertheless, we try to use it by drawing intuition from known rules of thumb from CNF encodings for SAT.

The translation of answer set programs to CNFs usually proceeds in two steps. First the

cyclic positive dependencies in the program need to be taken care of in *Cycle Breaking*. Only then, in the second step, Clark's Completion [Fag94] is guaranteed to produce a CNF that correctly captures the semantics of the program. For Clark's Completion itself, we consider it rather unlikely that there is a way to significantly reduce the size or semantic complexity of the resulting CNF. However, previous work by Hecher [Hec22] has shown that we can modify the completion to limit the treewidth increase compared to the *primal treewidth* of the program to a low constant factor. We improve this result to use the *incidence treewidth* of the program [JPW09], which provides bounds that are better or equal.

As for Cycle Breaking, we again need a more in depth analysis. There already exists a wide range of approaches [JN11; Jan04; MJ10; LZ03; Hec22]. We need to discard some of them, since they do not preserve answer sets in a bijective fashion [LZ03; Hec22] and examine encoding properties of the rest. Here, we see that while the "semantic complexity" of Janhunen and Niemelä's [JN11] work may prove to be problematic, due to the use of binary counters, its asymptotic guarantees for CNF size and treewidth are far better than that of Mantadelis and Janssens' [MJ10] work. Nevertheless, previous empirical results showed that the asymptotic guarantees do not necessarily translate to better performance, presumably due to high constant factors hidden by the BigO notation and the high "semantic complexity" [Fie+11]. We thus introduce a novel strategy for Cycle Breaking called T$_\mathcal{P}$-Unfolding, where we aim to reach a middle ground between the previous approaches. That is, we obtain similarly low "semantic complexity" and constant factors as Mantadelis and Janssens' [MJ10] approach but only moderately higher asymptotic size and treewidth guarantees than Janhunen and Niemelä's [JN11] approach. In order to achieve this, we resort not only to the number of atomic formulas that are involved in cyclic dependencies but a novel more sensitive measure of the cyclicity of the positive dependencies, related to the notions of *backdoors* into acyclicity [FS15] that intuitively asks how many atoms we would need to remove to obtain an acyclic (dependency) graph.

Apart from our theoretical advancements, we implement our findings in the open source AASC solver aspmc and use it to verify that our theoretical considerations indeed result in an increased performance in practice. For this, we conduct a thorough empirical evaluation on typical benchmarks from the probabilistic logic programming community, which confirms previous expectations regarding the CNF sizes and treewidths as well as the performance increase.

In the following, we first introduce the necessary preliminaries, in Section 4.1, and AASC, in Section 4.2. After this, we shortly go over some instantiations of AASC highlighting its broad applicability for quantitative inference, in Section 4.3, before directing our attention to AASC evaluation, in Section 4.4, giving special attention to different approaches for the Knowledge Compilation step. We finish the theoretical considerations with a study of Clark's Completions and Cycle Breakings, in Section 4.5 and Section 4.6, respectively. Based on this, we outline the practical realization in our solver aspmc, in Section 4.7, which we evaluate empirically, in Section 4.8. Finally, we conclude and highlight possible

further points of attack for improved performance, in Section 4.9.

4.1 Preliminaries

We recall the background on ASP, graph representations of programs, and their associated structural parameters.

4.1.1 Logic Programming

A (normal) *answer set program* Π is a finite set of *rules* r of the form

$$a \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n,$$

where a and all b_j, c_k are propositional atoms. Given such a rule r, we let

$$H(r) = a, \qquad B^+(r) = \{b_1, \ldots, b_n\},$$
$$B^-(r) = \{c_1, \ldots, c_n\}, \qquad B(r) = \{b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_n\}.$$

We slightly abuse notation and write

$$\leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n$$

for

$$\bot \leftarrow b_1, \ldots, b_m, \text{not } c_1, \ldots, \text{not } c_n, \text{not } \bot,$$

where $\bot$ is a propositional atom that otherwise does not occur in Π.

Furthermore, we allow *choice rules* $\{a\} \leftarrow B^+(r), B^-(r)$ as a shorthand for the two rules $a \leftarrow B^+(r), B^-(r), \text{not } na$ and $na \leftarrow B^+(r), B^-(r), \text{not } a$, where na is a fresh propositional atom. We denote by $\mathcal{A}(\Pi)$ the set of propositional atoms that occur in Π.

An *interpretation*, denoted $\mathcal{I}$, is a subset of $\mathcal{A}(\Pi)$; it *satisfies* an atom $a \in \mathcal{A}(\Pi)$ (resp. literal $\text{not } a$ for $a \in \mathcal{A}(\Pi)$), written $\mathcal{I} \models a$ (resp. $\mathcal{I} \models \text{not } a$), if $a \in \mathcal{I}$ (resp. $a \notin \mathcal{I}$). It satisfies Π (is a *model* of Π), if for each rule $r \in \Pi$ it holds that either $\mathcal{I} \models H(r)$ or $\mathcal{I} \not\models B(r)$, i.e., there exists some $l \in B(r)$ such that $\mathcal{I} \not\models l$. Furthermore, $\mathcal{I}$ is an *answer set* of Π if it is a $\subseteq$-minimal model of the *reduct*[1] $\Pi^{\mathcal{I}} = \{r \in \Pi \mid B^+(r) \subseteq \mathcal{I}, B^-(r) \cap \mathcal{I} = \emptyset\}$, of Π with respect to $\mathcal{I}$,i.e., the set of rules r in Π where $B(r)$ is satisfied. We denote the set of answer sets of a program Π by $\mathcal{AS}(\Pi)$.

As usual the schematic rules with variables $X, Y, \ldots$ are implicitly universally quantified and their semantics is given by grounding (instantiation) with concrete values (constants).

Example 39 (Smokers)**.** *We consider the smokers program, which is a standard example from probabilistic logic programming [DKT07].*

$$\{stress(X)\} \leftarrow person(X)$$

[1] All our results hold for both the FLP-reduct [FPL11] and GL-reduct [GL88]

$$\begin{aligned} \mathit{smokes}(X) &\leftarrow \mathit{stress}(X) \\ \{\mathit{inf}(X,Y)\} &\leftarrow \mathit{friend}(X,Y) \\ \mathit{smokes}(Y) &\leftarrow \mathit{smokes}(X), \mathit{inf}(X,Y) \end{aligned}$$

This encodes that for each person it is randomly determined whether they are stressed. Stressed persons smoke may influence friends, which is again random, to also smoke. We shall abbreviate stress(.) and smokes(.) by st(.) and sm(.), respectively.

Propositional Logic We aim to translate programs into formulas of propositional logic.

We use propositional formulas in Conjunctive Normal Form (CNF). A CNF $\mathcal{C}$, defined for a set V of variables, is a finite conjunction of *clauses* C_i, where each clause consists of a finite disjunction of *literals* $l \in \{v, \neg v\}$ for some $v \in V$.

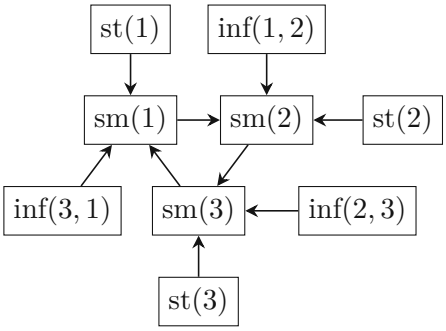
We use the standard satisfaction relation and call an interpretation $\mathcal{I} \subseteq V$ that satisfies a formula a *model*.

Graphs and Digraphs Especially in Section 4.6.4 we will consider graphs and digraphs, using the following notation. The vertex- and edge-set of a (di)graph $G = (V, E)$ is denoted by $V(G)$ and $E(G)$, respectively. For $V \subseteq V(G)$ we let $G[V]$ be the (di)graph obtained by removing all vertices not in V from $V(G)$ (i.e. $V(G[V]) = V(G) \cap V$) and removing all edges which use a vertex not in V (i.e. $E(G[V]) = E(V) \cap V \times V$). Further, we define $G \setminus V$ as $G[V(G) \setminus V]$. The subgraph $C = G[V]$ is *strongly connected* if every vertex in C is reachable from any other vertex in C. We denote by $\mathrm{SCC}(G)$ the set of *strongly connected components (SCC)* of G, which are strongly connected subgraphs $G[V]$, where V is subset-maximal.

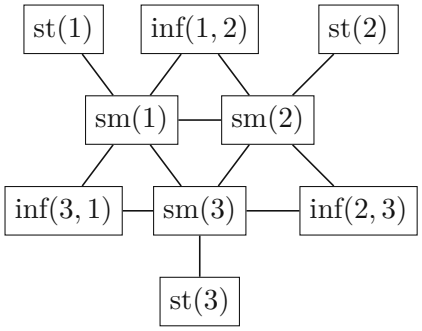
The (positive) *dependency graph* $\mathrm{DEP}(\Pi)$ of a program Π is the digraph G with $V(G) = \mathcal{A}(\Pi)$ and $(b, a) \in E(G)$ if there is a rule $r \in \Pi$ such that $a \in H(r)$ and $b \in B^+(r)$. The *primal graph* $\mathrm{PRIM}(\Pi)$ of Π is the graph G with $V(G) = \mathcal{A}(\Pi)$ and $\{x, y\} \in E(G)$ if there is a rule $r \in \Pi$ such that $x, y \in \{H(r)\} \cup B^+(r) \cup B^-(r)$. The *incidence graph* $\mathrm{INC}(\Pi)$ of Π is the graph G with $V(G) = \mathcal{A}(\Pi) \cup \Pi$ and $\{a, r\} \in E(G)$ if there is a rule $r \in \Pi$ such that $a \in \{H(r)\} \cup B^+(r) \cup B^-(r)$.

Example 40 (cont'd)**.** *Given the input data person(i) for $i = 1, \ldots, 3$ as well as friend(i, j) for $(i, j) = (1, 2), (2, 3), (3, 1)$, we can ground and reduce (by omitting facts and removing them from the bodies of rules, etc.) the smokers program to Π_{sm}*

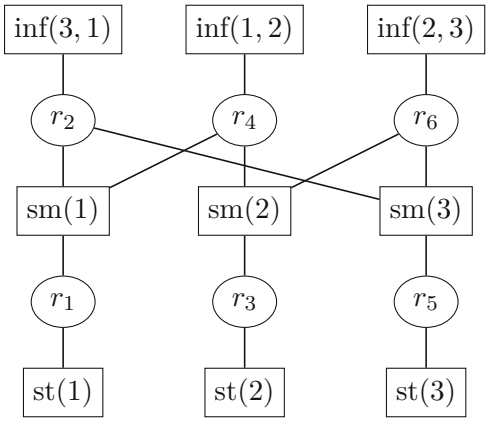
$$\begin{aligned} \{\mathit{st}(1)\} \leftarrow \quad \{\mathit{st}(2)\} \leftarrow \quad \{\mathit{st}(3)\} \leftarrow \\ \{\mathit{inf}(3,1)\} \leftarrow \quad \{\mathit{inf}(1,2)\} \leftarrow \quad \{\mathit{inf}(2,3)\} \leftarrow \\ r_1 = \mathit{sm}(1) \leftarrow \mathit{st}(1) \qquad r_2 = \mathit{sm}(1) \leftarrow \mathit{inf}(3,1), \mathit{sm}(3) \\ r_3 = \mathit{sm}(2) \leftarrow \mathit{st}(2) \qquad r_4 = \mathit{sm}(2) \leftarrow \mathit{inf}(1,2), \mathit{sm}(1) \\ r_5 = \mathit{sm}(3) \leftarrow \mathit{st}(3) \qquad r_6 = \mathit{sm}(3) \leftarrow \mathit{inf}(2,3), \mathit{sm}(2) \end{aligned}$$



(a) Dependency graph of Π_{sm} restricted to non-choice rules.



(b) Primal graph of Π_{sm} restricted to non-choice rules.



(c) Incidence graph of Π_{sm} restricted to non-choice rules. Vertices for rules are circles, vertices for atoms are rectangles

Figure 4.1: Different graphs associated with the running example program Π_{sm} for simplicity restricted to non-choice rules (i.e., with rules of the form $\{\text{stress}(x)\} \leftarrow$ etc.).

The dependency, primal, and incidence graph of the resulting program Π_{sm} without choice rules is given in Figure 4.1a, Figure 4.1b, and Figure 4.1c, respectively.

Next, we recall the definition of treewidth.

Definition 125 (Tree Decomposition, Treewidth)**.** *A* tree decomposition *of a graph G is a pair (T, χ), where T is a tree and χ is a labeling of $V(T)$ by subsets of $V(G)$ s.t.*

- *for all nodes $v \in V(G)$ there is $t \in V(T)$ s.t. $v \in \chi(t)$;*
- *for every edge $\{v_1, v_2\} \in V(E)$ there exists $t \in V(T)$ s.t. $v_1, v_2 \in \chi(t)$;*
- *for all nodes $v \in V(G)$ the set of nodes $\{t \in V(T) \mid v \in \chi(t)\}$ forms a (connected) subtree of T.*

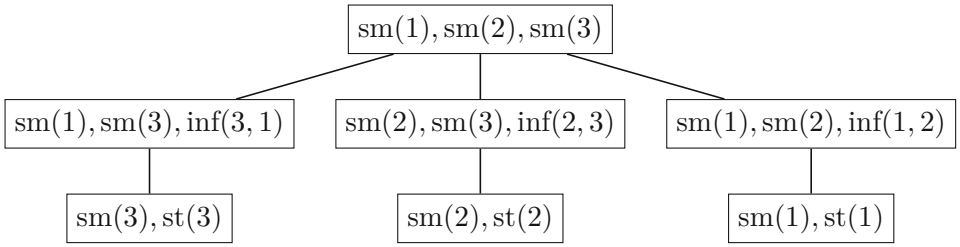


Figure 4.2: An optimal tree decomposition of the graph in Figure 4.1b. Each vertex is labeled by the vertices in the corresponding bag.

The width of (T, χ) *is* $\max_{t \in V(T)} |\chi(t)| - 1$. *The* treewidth *of a graph is the minimal width of any of its tree decompositions.*

Intuitively, treewidth is a measure of the distance of a graph from a tree. It is motivated by the fact that many computationally hard problems are tractable on trees. Correspondingly, trees are the only graphs that have treewidth 1. Given low treewidth it is then often possible to generalize tractability results by decomposing problems recursively into smaller subproblems using a tree decomposition witnessing the low width.

A more restricted parameter than treewidth is pathwidth.

Definition 126 (Path Decomposition, Pathwidth)**.** *Let* G *be a graph. Then a* path decomposition *is a tree decomposition* (T, χ)*, where* T *is a path.*

The pathwidth *of a graph is the minimal width of any of its path decompositions.*

Pathwidth is more restricted than treewidth but has similar properties, i.e., it allows for the generalization of tractability results on graphs that are paths. Notably, when the treewidth of an n-vertex graph G is k, then the pathwidth of G is in $\mathcal{O}(k \cdot \log(n))$ [Bod98].

Then the primal (resp. incidence) treewidth of a program is the treewidth of its primal (resp. incidence graph). The applies to pathwidth.

Example 41 (cont'd)**.** *The treewidth of the graph in Figure 4.1b is 2, as its tree decomposition in Figure 4.2 which has width 2 shows. A smaller width is not possible here, since the graph contains a clique over three vertices (e.g.* $sm(1), sm(2), sm(3)$*). The treewidth of the graph in Figure 4.1c is also 2, since it is not a tree as the cycle* $r_2 \to sm(3) \to r_6 \to sm(2) \to r_4 \to sm(1) \to r_2$ *shows.*

4.2 Algebraic Answer Set Counting

We now reintroduce ASP with algebraic measures as a basic formalism for Algebraic Answer Set Counting (AASC).

We use a variant of weighted logics [DG07] restricted to propositional formulas.

Definition 127 (Weighted Logic)**.** *Let $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ be a semiring. A* weighted formula *α over $\mathcal{R}$ is of the form*

$$\alpha ::= k \mid v \mid \neg v \mid \alpha + \alpha \mid \alpha * \alpha$$

where $k \in R$ and v is a propositional atom. The semantics of α w.r.t. an interpretation $\mathcal{I}$, denoted $[\![\alpha]\!]_\mathcal{R}(\mathcal{I})$, is

$$\begin{aligned}
[\![k]\!]_\mathcal{R}(\mathcal{I}) &= k, \\
[\![v]\!]_\mathcal{R}(\mathcal{I}) &= \begin{cases} e_\otimes & v \in \mathcal{I} \\ e_\oplus & \text{otherwise} \end{cases}, \\
[\![\neg v]\!]_\mathcal{R}(\mathcal{I}) &= \begin{cases} e_\oplus & v \in \mathcal{I} \\ e_\otimes & \text{otherwise} \end{cases}, \\
[\![\alpha_1 + \alpha_2]\!]_\mathcal{R}(\mathcal{I}) &= [\![\alpha_1]\!]_\mathcal{R}(\mathcal{I}) \oplus [\![\alpha_2]\!]_\mathcal{R}(\mathcal{I}), \\
[\![\alpha_1 * \alpha_2]\!]_\mathcal{R}(\mathcal{I}) &= [\![\alpha_1]\!]_\mathcal{R}(\mathcal{I}) \otimes [\![\alpha_2]\!]_\mathcal{R}(\mathcal{I}).
\end{aligned}$$

We can use weighted formulas to express calculations over a semiring, depending on the truth values of atoms with respect to an interpretation.

Example 42 (cont'd)**.** *As mentioned before, the smokers program is a typical example from the probabilistic domain. Using weighted formulas, we can introduce probabilities. We define*

$$\alpha = \Pi_{i=1}^{3} st(i) * 0.4 + \neg st(i) * 0.6 \tag{4.1}$$

$$* \Pi_{i,j=1,2,3,i+1 \equiv j \bmod 3} inf(i,j) * 0.3 + \neg inf(i,j) * 0.7. \tag{4.2}$$

The first line 4.1 tells us that every person i is either stressed, with probability 0.4*, or not stressed, with probability* 0.6*. With the second line (4.2) we capture that for $(i,j) = (1,2)$, $(2,3)$, $(3,1)$ person i is influenced by person j with probability* 0.3 *and not influenced with probability* 0.7*.*

Using weighted formulas, we define algebraic measures as follows:

Definition 128 (Algebraic Measure)**.** *An* algebraic measure *$\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$ consists of an answer set program Π, a weighted formula α, and a semiring $\mathcal{R}$. Then, the weight of an answer set $\mathcal{I} \in \mathcal{AS}(\Pi)$ under μ is defined by*

$$\mu(\mathcal{I}) = [\![\alpha]\!]_\mathcal{R}(\mathcal{I}).$$

Additionally, the result of an (atomic) query *for an atom $a \in \mathcal{A}(\Pi)$ is given by*

$$\mu(a) = \bigoplus_{\mathcal{I} \in \mathcal{AS}(\Pi), a \in \mathcal{I}} \mu(\mathcal{I}),$$

and the result of the overall weight query *of Π is*

$$\mu(\Pi) = \bigoplus_{\mathcal{I} \in \mathcal{AS}(\Pi)} \mu(\mathcal{I}).$$

Example 43 (cont'd)**.** *Using algebraic measures, we perform probabilistic reasoning. We combine the weighted formula α, which handles the probabilities, and the program Π_{sm}, which handles the logical background theory, to the measure $\mu_{sm} = \langle \Pi_{sm}, \alpha, \mathcal{P} \rangle$. Then the answer set $\mathcal{I} = \{st(1), sm(1)\}$ has weight $\mu_{sm}(\mathcal{I}) = 0.4 \cdot 0.6^2 \cdot 0.7^3$.*

The query $\mu(sm(1))$ corresponds to the probability that $sm(1)$ holds. To evaluate it, we need to perform AASC, i.e., sum up the probabilities of all answer sets s.t. $sm(1)$ holds. Since Π_{sm} has 2^6 answer sets out of which $2^5 + 2^3 + 2$ include $sm(1)$, we refrain from computing $\mu(sm(1))$ naïvely by considering all the answer sets separately. Instead, we consider the following three disjoint cases in which $sm(1)$ holds:

1. *$st(1)$ is true and the other probabilistic atoms take an arbitrary truth value;*
2. *$st(1)$ is false, $inf(3,1)$ and $st(3)$ are true and the other probabilistic atoms take an arbitrary truth value;*
3. *$st(1)$ and $st(3)$ are false, $inf(3,1)$, $inf(2,3)$ and $st(2)$ are true, and $inf(1,2)$ takes an arbitrary truth value.*

These three cases are exclusive, i.e., no answer set falls in any two of the cases, and furthermore, they cover all the answer sets in which $sm(1)$ is true. Thus, we can compute the probability of $sm(1)$ as a sum of the probabilities of the three cases. These are $0.4, 0.6 * 0.3 * 0.4 (= 0.072)$ and $0.6 * 0.6 * 0.3 * 0.3 * 0.4 (= 0.01296)$, respectively, since it is enough to take the product of the probabilities of the atoms whose truth values we assert as the other atoms that are left arbitrary only contribute a factor of 1. Thus, $\mu(sm(1)) = 0.48496$.

Following the conventions of [BD20], we also introduce factorized measures.

Definition 129 (Factorized Measure)**.** *Let $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$ be an algebraic measure and $F \subseteq \mathcal{A}(\Pi)$. Then μ is* factorized *w.r.t. F, if there is a weight function $\beta : F \cup \{\neg f \mid f \in F\} \to R$ s.t. for all $\mathcal{I} \in \mathcal{AS}(\Pi)$ it holds that*

$$\mu(\mathcal{I}) = \bigotimes_{f \in F \cap \mathcal{I}} \beta(f) \otimes \bigotimes_{f \in F \setminus \mathcal{I}} \beta(\neg f).$$

The difference between factorized and non-factorized measures is intuitively how complicated the weight function is. The former must be expressible as a product of weights of literals that are true in a given interpretation, whereas the latter allow complex arithmetic expressions using both the addition and multiplication of weights in dependence on the true literals in the interpretation.

The motivation behind considering factorized measures is that current frameworks, such as ProbLog and algebraic Prolog only use factorized measures. This raises the question of whether this covers everything we need.

Example 44 (cont'd). *The measure μ_{sm} is factorized over $st(i), i = 1, 2, 3$ and $inf(i, j), i+1 \equiv j \bmod 3$, by letting $\beta(st(i)) = 0.4, \beta(\neg st(i)) = 0.6$ and $\beta(inf(i, j)) = 0.3, \beta(\neg st(i, j)) = 0.7$.*

Not every measure is factorized however.

Example 45 (Non-factorized). *For an example of a non-factorized measure, consider the measure $\mu_w = \langle \Pi, a + b + (-1 * \neg a * \neg b), \mathbb{Z} \rangle$. It has value 2 if both a and b hold, 1 if one of them holds and -1 otherwise. This measure is not factorized over $F = \{a, b\}$ as there are no values $\beta(a), \beta(b), \beta(\neg a), \beta(\neg b) \in \mathbb{Z}$, s.t.*

$$\beta(a) \cdot \beta(b) = 2 \qquad \beta(a) \cdot \beta(\neg b) = 1$$
$$\beta(\neg a) \cdot \beta(b) = 1 \qquad \beta(\neg a) \cdot \beta(\neg b) = -1$$

Note that there are also no such values in $\mathbb{R}$.

While not every algebraic measure is factorized, there always exists a factorized measure that preserves weights of queries. To establish this, we need some notation for sets of indexed subformulas of a weighted formula α.

Definition 130 (Subformulas). *Let α be a weighted formula. Then $\mathcal{S}(\alpha)$ is the set of pairs (i, β), where β is a subformula of α indexed by a position-string $i \in \{0, 1\}^*$ in the syntax-tree of α, that is:*

- *For $\alpha \in \{p(\vec{x}), \neg p(\vec{x}), k\}$ we let $\mathcal{S}(\alpha) = \{(\epsilon, \alpha)\}$.*
- *For $\alpha \in \{\alpha_1 + \alpha_2, \alpha_1 * \alpha_2\}$ we let $\mathcal{S}(\alpha) = \{(\epsilon, \alpha)\} \cup \{(0r, \beta) \mid (r, \beta) \in \mathcal{S}(\alpha_1)\} \cup \{(1r, \beta) \mid (r, \beta) \in \mathcal{S}(\alpha_2)\}$.*

Theorem 131 (Factorization). *Let $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$ be an algebraic measure. Then we can construct a factorized algebraic measure $\mu' = \langle \Pi', \alpha', \mathcal{R} \rangle$ s.t. for $a \in \mathcal{A}(\Pi) : \mu(a) = \mu'(a)$ in linear time.*

Proof (sketch). Intuitively, we implicitly apply the distributive law. For this, we introduce a new atom α_i for every indexed subformula $(i, \beta) \in S(\alpha)$, which is true if the value of the subformula β at index i is included in the current product. To implement this, we let $\Pi' = \Pi \cup \Pi_{\text{root}} \cup \Pi_* \cup \Pi_+ \cup \Pi_{\text{leaf}}$, where $\Pi_{root} = \{\leftarrow \text{not } \alpha_\epsilon\}$ ensures that the value of the formula at the root is included.

$$\Pi_* = \left\{ \begin{array}{l} \alpha_i \leftarrow \alpha_{i.0}, \alpha_{i.1} \\ \leftarrow \alpha_{i.0}, \text{not } \alpha_{i.1} \\ \leftarrow \text{not } \alpha_{i.0}, \alpha_{i.1} \end{array} \middle| \; (i, \beta_1 * \beta_2) \in \mathcal{S}(\alpha) \right\}$$

ensures that a subformula that uses multiplication is only included if both subformulas are included. Further, we ensure that either both or none of the subformulas are included.

$$\Pi_{+} = \left\{ \begin{array}{c} \alpha_i \leftarrow \alpha_{i.1} \\ \alpha_i \leftarrow \alpha_{i.0} \\ \leftarrow \alpha_{i.0}, \alpha_{i.1} \end{array} \middle| \; (i, \beta_1 + \beta_2) \in \mathcal{S}(\alpha) \right\}$$

ensures that a subformula that uses addition is only included if one of the subformulas is included. Furthermore, we ensure that at most one of the subformulas is included.

$$\begin{aligned} \Pi_{\text{leaf}} = &\{\{\alpha_i\} \leftarrow a \mid (i, a) \in \mathcal{S}(\alpha)\} \cup \\ &\{\{\alpha_i\} \leftarrow \text{not } a \mid (i, \neg a) \in \mathcal{S}(\alpha)\} \cup \\ &\{\{\alpha_i\} \leftarrow \; \mid (i, k) \in \mathcal{S}(\alpha), k \in R\}. \end{aligned}$$

Formally, we define

$$\alpha' = \Pi_{(i,k) \in \mathcal{S}(\alpha), k \in R} \;\; \alpha_i * k + \neg \alpha_i.$$

Then $\mu' = \langle \Pi', \alpha', \mathcal{R} \rangle$ is factorized, by choosing $F = \{\alpha_i \mid (i, k) \in \mathcal{S}(\alpha), k \in R\}$ and $\beta(\alpha_i) = k, \beta(\neg \alpha_i) = e_{\otimes}$.

Furthermore, along a similar line of reasoning as in [EK21] it follows that $\mu(\Pi) = \mu'(\Pi')$. □

Hence, any AASC instance given as an algebraic measure can be reduced to AASC for a factorized measure. Thus, we can focus on factorized measures.

4.3 Applications

We covered the introductory theory necessary when dealing with AASC and algebraic measures. In the following, we shortly reiterate some of the prominent applications of AASC via instantiation with fixed semirings.

The idea behind AASC is ubiquitous in quantitative reasoning: one takes a Boolean logic and turns it into a weighted logic, by replacing conjunctions by multiplications and disjunctions by additions over a semiring. Other such semiring reasoning frameworks are for example Semiring-based Constraint Satisfaction Problems [Bis+99], provenance semantics for datalog [GKT07], and Algebraic Model Counting (AMC) [KVD17], to name only a few. Due to this, a wide range of applications of the semiring semantics are already known. Kimmig, Van den Broeck, and De Raedt [KVD17] give an extensive list of different reasoning tasks that can be performed using the semiring semantics and discuss the appropriate semirings for it.

We therefore do not extensively list all the different possibilities here but focus on some that relate to existing extensions of ASP and, thus, have higher relevance for our setting. Namely, we focus on two different classes of problems and touch upon a third. First, we consider problems that are #P-hard, i.e., at least as hard as counting the number

of satisfying assignments of a propositional formula. For these problems knowledge compilation is a well-established and competitive technique [Dar04; OD15]. Second, when we work over idempotent semirings, algebraic measure evaluation is often equivalent to OptP-hard problems. Here, at least in the area of Weighted MaxSAT, a prototypical OptP-hard problem, knowledge compilation does not seem to be a good strategy, given that none of the solvers of the recent MaxSAT Evaluation [Bac+20] is knowledge compilation based. Thus, while we relate AASC to OptP-hard problems here and cover their solution also with our implementation (which works for general commutative semirings) we do not expect performance improvements compared to specialized tools such as clingo [Geb+14] here.

We note that there are relevant problems that are in a complexity class above #P. Recent work [KTK22] showed that many such problems can be solved by AASC over two semirings but not over one of them. Here, the problems are typically harder than AASC over one semiring, since they remain NP-hard on the usual tractable circuit representations [OCD16b]. Note that as with the OptP-hard problems, an implementation that works for general general semirings is available but not further considered here, since we focus on the evaluation of problems that are #P-hard but solvable on typical tractable circuit representations. For details we refer the interested reader to [KTK22].

4.3.1 #P-hard Problems

Probabilistic Reasoning The most common application that involves weighted model counting over the answer sets, in the typical sense, is probabilistic inference as in ProbLog, LP^{MLN}, or P-log. The specification and semantics of probabilistic programs as well as the allowed class of programs varies between the languages.

The main difference is that ProbLog uses Sato's distribution semantics [Tai95], which assumes that the only non-determinism in the program is due to a set of probabilistic facts and that the program has no constraints. This means that any given truth assignment to the probabilistic facts can be *uniquely* extended to a stable model of the program. The benefit of this restriction is that the program directly specifies a probability measure, meaning the sum of the weights of all answer sets adds up to one. On the other hand, LP^{MLN} and P-log do not impose such a restriction, thus allowing for a broader class of programs. Here, in order to obtain a probability measure, the probability of an answer set is not its weight but its *normalized* weight. That is, its weight is divided by the sum of the weights of all answer sets.

Apart from that, LP^{MLN} features another property. Namely, while ProbLog and P-log only allow for weights of atoms or rules that are between 0 and 1, LP^{MLN} allows for any real number as a weight.

For probabilistic inference by means of AASC, both of these differences are not significant. We can perform probabilistic inference in any of the three languages by evaluating an algebraic measure over an answer set program. For ProbLog programs, it is sufficient to

evaluate $\mu(a)$ for the algebraic measure μ corresponding to the program to obtain the probability that the query atom a is true according to the program. In the ProbLog case, we can use the probabilistic semiring $\mathcal{P}$ since only weights in $[0, 1]$ occur. For LP^{MLN} and P-log programs, we can compute the probability of a query atom a by evaluating $\mu(a)$ and $\mu(\text{not } a)$ over the semiring $\mathbb{R}$ of the real numbers. Then $Pr(a)$, the probability of a, is $\mu(a)/\mu(a)+\mu(\text{not } a)$. For a closer look at the relationship of the different probabilistic logic programming languages we refer to [RS18; LY17; Hah+22].

Neuro-Symbolic Reasoning Neuro-symbolic reasoning, as for example in DeepProbLog [Man+21], takes the idea of probabilistic reasoning with programs one step further. Here, instead of assuming that the probabilities of rules and facts are given, they are determined by a machine learning model that predicts the probability of a fact or rule for a given ground instantiation of it. Instead of having the same probability of stress for each person

$$0.4 :: \text{stress}(X) : - \ \text{person}(X).$$

we might have a *neural predicate* nn-stress($Age, Employment$) that models the probability of a person being stressed based on age and employment. In the DeepProbLog language, this would be expressed as

$$\text{nn-stress}(A, E) :: \text{stress} : - \ \text{person}(X), \text{age}(A, X), \text{employment}(E, X).$$

We consider two tasks in neuro-symbolic reasoning that need to be solved.

(T1) Determine the probability of a query, given a neuro-symbolic program.

For DeepProbLog, this again corresponds to a query using an algebraic measure: when the machine learning model that predicts the probabilities of facts and rules is *fixed* we can see DeepProbLog programs as ProbLog programs and, thus, evaluate probabilistic queries using algebraic measures. Here, we can again use the probabilistic semiring $\mathcal{P}$.

(T2) Train the model to predict the correct probabilities.

Usually, this is done using gradient descent to minimize some error function *err* that depends on the output of the neural networks. In order to use gradient descent, it is necessary to compute the derivative of *err*. In the DeepProbLog setting, this can be rather complicated, as *err* is not limited to measuring whether the predicted value was correct. Instead, the error can also be determined by the probability of a derived atom. E.g. in the smokers setting, we may have data about whether a person smokes but not about a persons stress. Thus, applying gradient descent to the error function of the smokers model involves obtaining its derivative in terms of the output of the neural network.

Manhaeve, Dumancic, Kimmig, Demeester, and Raedt [Man+21] showed that we can compute this gradient by weighted model counting over the *gradient semiring GRAD*. Therefore, we can also model this task using an algebraic measure over

$$GRAD = ([0,1] \times \mathbb{R}, \oplus, \otimes, (0,0), (1,0)),$$

where

$$(p_1, d_1) \oplus (p_2, d_2) = (p_1 + p_2, d_1 + d_2)$$
$$(p_1, d_1) \otimes (p_2, d_2) = (p_1 p_2, d_1 p_2 + d_2 p_1).$$

Intuitively, the first entry corresponds to the probability, as before, whereas the second entry computes the derivative of the first with respect to some parameter.

Knowledge Compilation Using the AASC approach we may further obtain a tractable circuit representation of the program. This may seem counterintuitive, since we said in the Introduction that we use knowledge compilation to perform AASC and not the other way around. However, it shows that when we can do weighted model counting over semirings, then we can also do knowledge compilation.

For knowledge compilation, we need a somewhat non-standard semiring similar to the provenance semirings of Green, Karvounarakis, and Tannen [GKT07]. Given a semiring $\mathcal{R}$ and a set of propositional atoms $\mathcal{V}$, we define the knowledge compilation semiring of algebraic circuits over $\mathcal{V}$ and $\mathcal{R}$ as

$$\mathcal{KC}(\mathcal{V}, \mathcal{R}) = (\mathcal{AC}(\mathcal{V}, \mathcal{R}), +, *, \bot, \top)$$
$$\mathcal{AC}(\mathcal{V}, \mathcal{R}) = \{\alpha \mid \alpha \text{ is an algebraic circuit over } \mathcal{V} \text{ and } \mathcal{R}\}$$
$$\alpha + \beta = \alpha + \beta$$
$$\alpha * \beta = \alpha * \beta$$

Here, an algebraic circuit over $\mathcal{V}$ and $\mathcal{R}$ is a directed acyclic graph (DAG), where each vertex is labeled by either $+, *$, a literal $l \in \mathrm{Lit}(\mathcal{V})$ with $\mathrm{Lit}(\mathcal{V}) = \mathcal{V} \cup \{\neg v \mid v \in \mathcal{V}\}$, or a semiring value $r \in R$, in such a manner that vertices without incoming edges are either labeled by a literal or a semiring value and vertices with incoming edges are either labeled by $+$ or $*$. Furthermore, there must be exactly one vertex that does not have outgoing edges, which we refer to as the *output*.

Clearly, using the standard equality in the sense of syntactical equality is inadequate, since $\mathcal{KC}(\mathcal{V}, \mathcal{R})$ is not commutative:

$$a + b \neq_{\text{syntax}} b + a.$$

We instead use semantic equality, i.e., two algebraic circuits $\alpha, \beta \in \mathcal{AC}(\mathcal{V}, \mathcal{R})$ are *semantically equal* if for every weight assignment $\sigma : \mathrm{Lit}(\mathcal{V}) \to R$ it holds that

$$[\![\alpha[\sigma]]\!]_{\mathcal{R}} = [\![\beta[\sigma]]\!]_{\mathcal{R}},$$

Here, $\alpha[\sigma]$ denotes that we replace every literal l in α by $\sigma(l)$ its semiring value under σ. Then $[\![\alpha[\sigma]]\!]_{\mathcal{R}}$ corresponds to evaluating the circuit by inductively defining the value of a node with a semiring label r as r and the value of a node labeled by $+$ (resp. $*$) with incoming nodes with values $r_1, \ldots, r_n$ as $\bigoplus_{i=1}^{n} r_i$ (resp. $\bigotimes_{i=1}^{n} r_i$. The final value of $[\![\alpha[\sigma]]\!]_{\mathcal{R}}$ is then given by the value of the output of the circuit.

Note that this idea is not new: Kimmig, Van den Broeck, and De Raedt [KVD11] introduced a BDD semiring, where conjunction and disjunction correspond to the respective operation on BDDs, thus, allowing knowledge compilation to a BDD. However, in contrast to the BDD semiring, our semiring allows any kind of algebraic circuit, even those that classically do not correspond to a tractable circuit class such as BDD, SDD, or d-DNNF.

Apart from this, we may obtain syntactically different results depending on how we evaluate a query over $\mathcal{KC}(\mathcal{V}, \mathcal{R})$. This, however, is of no concern to us. We can show that given any such syntactic representation of the result, we can evaluate meaningful queries.

Lemma 132 (Semiring Knowledge Compilation)**.** *Let $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$ be a factorized algebraic measure with weight function β that factorizes μ. Furthermore, let*

$$\mu_{KC} = \langle \Pi, \alpha_{KC}, \mathcal{KC}(\mathcal{A}(\Pi), \mathcal{R}) \rangle$$
$$\alpha_{KC} = \Pi_{v \in \mathcal{A}(\Pi)} (v * \mathbf{v} + \neg v * \neg \mathbf{v}),$$

where we write atoms in italics v and semiring values in boldface $\mathbf{v}$.

Then given any algebraic circuit γ that represents the result of $\mu_{KC}(\Pi)$, we have that $\mu(\Pi) = [\![\gamma[\sigma_\beta]]\!]_{\mathcal{R}}(\emptyset)$, where $\sigma_\beta : Lit(\mathcal{A}(\Pi) \to R$ is given by $\sigma_\beta(l) = \beta(l)$.

Thus, given any algebraic circuit γ that corresponds to the result of the knowledge compilation query $\mu_{KC}(\Pi)$, we can evaluate any factorized measure query over $\mathcal{R}$ using γ in linearly many semiring operations in the size of γ. Thus, the key is to compute a small circuit γ and the latter efficiently. It is well-known that such arithmetic circuits can easily be obtained from BDDs, SDDs, or d-DNNFs [DD20], by replacing disjunctions and conjunctions by additions and multiplications, respectively.

For us the algebraic circuit semirings are not only of theoretical but also of practical interest. Namely, as we describe in more detail later in the section on implementation, we use SHARPSAT-TD [KJ21], which enables weighted model counting over semirings. Thus, we can use it for knowledge compilation using the idea of the algebraic circuit semirings. This again shows the power of the semiring paradigm. Since SHARPSAT-TD is implemented in such a manner that it works for general commutative semirings, we can solve a variety of different interesting tasks by performing a minor modification. In fact, the adaptation of SHARPSAT-TD to enable knowledge compilation to smooth d-DNNF was possible in less than 500 lines of code, including the parsing of additional input arguments and the adaptation of the preprocessor to handle weights over general semirings.

4.3.2 OptP-hard Problems

The applications discussed above share the fact that they are Weighted Model Counting problems in the strict sense, i.e. they are #P-hard. We thus expect that they are practically harder to solve using current implementations than other AASC problems which are not #P-hard. For example, AASC over the Boolean semiring $\mathbb{B}$ corresponds to checking the consistency of an answer set program and is therefore NP-complete. Arguably, it makes sense to consider such problems which are practically easier to solve separately.

Another class of problems that are affected by this reasoning are *optimization problems*. There is a broad variety of extensions of ASP that introduce different ways to add preferences to programs [Bre+15; BLR97]. We focus instead on theMost Probable Explanation (MPE) task from the probabilistic logic programming literature [Sht+14], which also fits into the context of optimization problems.

Most Probable Explanation The Most Probable Explanation (MPE) task is to determine the most likely probabilistic choices that led to a set of observed facts. We consider it in the setting of ProbLog programs.

Definition 133 (MPE)**.** *Given a ProbLog program* Π *and a set* $E = \{e_1, \ldots, e_n\}$ *of atoms called evidence, MPE is to determine the most likely assignment to the probabilistic facts such that* E *is entailed. That is compute*

$$p^* = \text{argmax}_{\mathcal{I} \in \mathcal{AS}(\Pi), E \subseteq \mathcal{I}} \prod_{p::f \in \Pi, f \in \mathcal{I}} p \cdot \prod_{p::f \in \Pi, f \notin \mathcal{I}} (1-p). \tag{4.3}$$

As we have seen before, we can define a measure μ over the probabilistic semiring $\mathcal{P}$ such that

$$\mu(\mathcal{I}) = \prod_{p::f \in \Pi, f \in \mathcal{I}} p \cdot \prod_{p::f \in \Pi, f \notin \mathcal{I}} (1-p).$$

Naturally, we could use μ as a function to optimize for modeling MPE. However, it would be desirable to have an algebraic measure whose overall weight corresponds to the solution of MPE.

Indeed, we can construct such a measure based on the max-times semiring $\mathcal{R}_{\max,\cdot}$. Recall that the multiplication of $\mathcal{R}_{\max,\cdot}$ corresponds to the usual product and that the addition corresponds to taking the maximum. Then, we can use the following formula α_L over $\mathcal{R}_{\max}$ to compute the likelihood, i.e. the probability of a given assignment to the probabilistic facts:

$$\Pi_{p::f \in \Pi}(f * p + \neg f * (1-p)).$$

Lemma 134 (MPE as AASC)**.** *Given a ProbLog program* Π *and evidence* E*, the overall weight of the algebraic measure* $\mu = \langle \Pi \cup \{ \texttt{:- } not\, e \mid e \in E\}, \alpha_L, \mathcal{R}_{\max,\cdot} \rangle$ *is equal to* p^*

in Equation (4.3)*, the maximum likelihood of any assignment to the probabilistic facts in* Π *such that* E *is entailed.*

Notably, this does not solve MPE strictly speaking, since we do not know the assignment that leads to the maximum probability. However, we can extend the max-times semiring $\mathcal{R}_{\max,\cdot}$ to perform bookkeeping that tracks which interpretation was used. For this, we use pairs $(p, \mathcal{I})$, where $p \in \mathbb{R}$ and $\mathcal{I}$ is a partial interpretation. Then addition of two such pairs $(p_i, \mathcal{I}_i), i = 1, 2$ takes that pair, where p_i is maximal, breaking ties with an arbitrary but fixed order on the partial interpretations. Multiplication of two such pairs results in $(p_1 \cdot p_2, \mathcal{I}_1 \cup \mathcal{I}_2)$. By changing the probabilistic facts f with weight p to have weight $(p, \{f\})$ instead, we obtain the desired output.

4.3.3 Harder Problems

There are also problems harder than #P, i.e., problems that stay hard on d-DNNFs without additional restrictions. Such problems frequently occur in probabilistic logic programming. Some examples are Maximum A Posteriori (MAP) problems, probabilistic inference using the semantics for general answer set programs of Totis, Kimmig, and Raedt [TKR23] and Skryagin, Stammer, Ochs, Dhami, and Kersting [Skr+22], and Maximum Expected Utility (MEU) problems [DD20; Van+10].

What all these problems share is that they come with a partition X_O, X_I of the variables and two semirings $\mathcal{R}_O, \mathcal{R}_I$ such that their final result can be expressed as

$$\bigoplus^{O}_{\mathcal{I}_O \subseteq X_O} \bigotimes^{I}_{l \in \text{lit}(X_O), \mathcal{I} \models l} \alpha_O(l) \otimes^{I} t\left(AASC(\mathcal{I}_O)\right), \text{ where} \tag{4.4}$$

$$AASC(\mathcal{I}_O) = \bigoplus^{I}_{\mathcal{I}_I \subseteq X_I, \mathcal{I}_O \cup \mathcal{I}_I \in \mathcal{AS}(\Pi)} \bigotimes^{I}_{l \in \text{lit}(X_I), \mathcal{I} \models l} \alpha_I(l), \tag{4.5}$$

for functions α_O and α_I that assign the literals over X_O and X_I a weight from $\mathcal{R}_O$ and $\mathcal{R}_I$, respectively, and a transformation function t that maps semiring values from $\mathcal{R}_I$ to $\mathcal{R}_O$. Intuitively, this means that we have to solve one AASC instance $AASC(\mathcal{I}_O)$ for each assignment $\mathcal{I}_O$ to the outer variables X_O and sum up the results, i.e., we need to perform nested AASC.

Consider for example the MAP problem. It is similar to MPE in the sense that both problems we ask for the most likely assignment to a set of variables X_O, given some evidence. However, while for MPE the probability of the most likely assignment is determined as the product of the probabilities of the assignments to the variables in X_O, in MAP this product is multiplied by the marginal probability over the remaining probabilistic variables in the set X_I. Thus, for MAP the expression in (4.5) corresponds to

$$\begin{aligned}&\text{argmax}_{\mathcal{I}_O \subseteq X_O} \prod_{l \in \text{lit}(X_O), \mathcal{I} \models l} \alpha_O(l) \cdot \sum_{\mathcal{I}_I \subseteq X_I, \mathcal{I}_O \cup \mathcal{I}_I \in \mathcal{AS}(\Pi)} \prod_{l \in \text{lit}(X_I), \mathcal{I} \models l} \alpha_I(l) \\ &= \text{argmax}_{\mathcal{I}_O \subseteq X_O} Pr(\mathcal{I}_O) \cdot Pr(E \mid \mathcal{I}_O),\end{aligned}$$

where $Pr(\mathcal{I}_O)$ and $Pr(E \mid \mathcal{I}_O)$ denote the probability of $\mathcal{I}_O$ and the probability of the evidence E given $\mathcal{I}_O$, respectively. The inner sum computes $Pr(E \mid \mathcal{I}_O)$, the conditional probability of E given the assignment $\mathcal{I}_O$ to the outer variables X_O and the outer sum takes the maximum over all assignments to the outer variables. Here, we assume that Π includes the constraints that the evidence should hold, as in Lemma 134.

Since all these problems are also definable algebraically and can be solved by compilation to *constrained* tractable circuits instead of arbitrary tractable circuits, the results of the rest of the paper are also relevant for this setting. Again, we see the flexibility of the algebraic approach that allows us to consider a large variety in the same manner. For more details, including the adaptations that are necessary in the knowledge compilation step to allow for *constrained* knowledge compilation we refer to [KTK22].

4.4 Solving AASC Problems

The main goal of our work is to solve AASC problems more efficiently. In order to do so, we need to aggregate the values associated with each answer set of a program by the weighted formula. Already computing one answer set is NP-hard and computing an aggregate of over all of them can be even harder, depending on the chosen aggregate, which in our setting corresponds to the addition of the semiring. There has, however, been a lot of research on different approaches that can be used to overcome this hardness in an effort to provide an efficient implementation [Fie+11; Fie+15; MJ10; Vla+16; RS11]. While these approaches differ significantly in their details, they usually follow a similar overall workflow.

In this section, we give a broad overview of both the shared workflow and the differences between the approach to arrive at a solution. Especially, we investigate the conditions under which each of the approaches is likely to succeed. This is interesting, on the one hand, from a user perspective, as it can allow one to make an informed choice for a tool, depending on the conditions satisfied by the instances at hand. On the other hand, it useful for us to make an informed decision which of the approaches we deem to have the most potential for AASC in general allowing us to use it as the basis for our work.

4.4.1 Overall Workflow

A schema of the general pipeline that is usually shared between different solvers is shown in Figure 4.3. Many steps are optional and different solvers use different paths through this pipeline. Usually, solvers accept possibly non-ground programs Π with some additional annotations for the algebraic measure. Often, this program is then grounded, resulting in a propositional program, which is simplified in the next step. Afterwards, the actual AASC starts, by converting the program into an equivalent tractable circuit representation $\mathcal{C}$ via some form of *knowledge compilation* (KC). In the last step, the result is obtained by evaluation over the circuit $\mathcal{C}$. Before we go into more details, we would

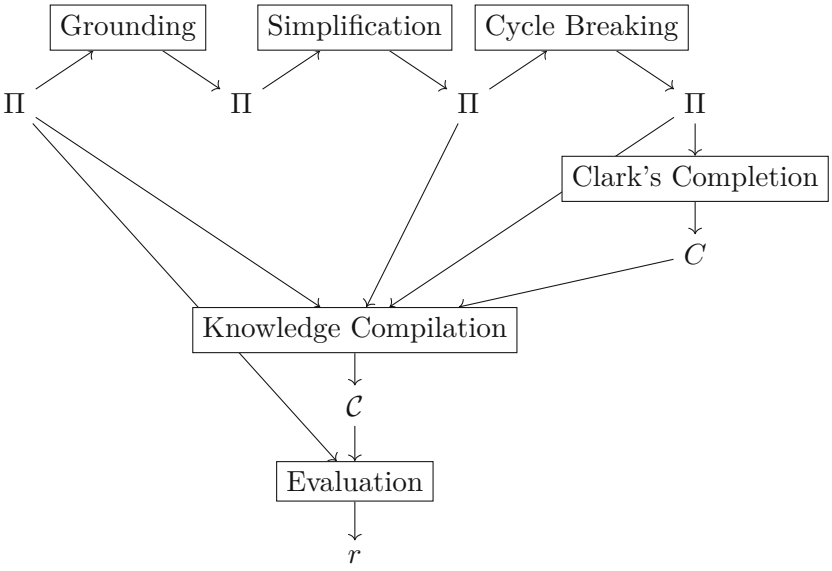


Figure 4.3: Schema of the overall workflow of existing solvers for the evaluation of AASC problems.

like to note that in some solvers these steps are intertwined instead of being executed one after the other.

We focus on the knowledge compilation step. First, there has already been a lot of work on the other steps of the pipeline: Fierens, Van den Broeck, Renkens, Shterionov, Gutmann, Thon, Janssens, and De Raedt [Fie+15] showed that for probabilistic inference with ProbLog programs, it is sufficient to restrict oneself to the relevant ground part of the program, which can significantly reduce the size of the problem. Tsamoura, Gutiérrez-Basulto, and Kimmig [TGK20] went even further and used the magic set technique [Ban+86] to identify the relevant part during grounding, thus, eliminating the need to even produce the remaining irrelevant part in this phase. Additionally, Shterionov [Sht15] [Sht15, Chapter 3] thoroughly investigated a wide range of simplifications for probabilistic logic programs that were shown to lead to significantly improved solving times on many benchmark sets.

Second, we argue that the knowledge compilation step has the largest potential for improvement and is likely to lead to the biggest effect. Due to these reasons, we focus on the knowledge compilation step, as we assume its investigation to be the most fruitful.

4.4.2 Knowledge Compilation

The idea behind knowledge compilation is the following: while certain problems, such as AASC, are hard on general logical theories, there are so called *tractable circuit representations*, where this is not the case. Before we discuss different possible strategies that are currently employed to compile programs into tractable circuit representations,

we recall the different circuit classes that are commonly used in the area of probabilistic logic programming.

A very simple circuit class is the MODS representation.

Definition 135 (MODS)**.** *A logical theory $\mathcal{T}$ over propositional variables $\mathcal{V}$ is a MODS theory, if it is of the form*

$$\bigvee_{\mathcal{I}\subseteq\mathcal{V},\mathcal{I}\models\mathcal{T}} \bigwedge_{v\in\mathcal{I}} v \wedge \bigwedge_{v\in\mathcal{V}\setminus\mathcal{I}} \neg v,$$

i.e., if it is a disjunction of its models, where each model is represented by the conjunction of its true literals.

Clearly, given a theory $\mathcal{T}$ in MODS representation tasks like probabilistic inference are possible in polynomial time in the size of the input $\mathcal{T}$. However, the example of MODS also has obvious downsides: converting a program Π into an equivalent propositional theory $\mathcal{T}$ in MODS representation can result in a theory whose size is exponential in the one of Π, since we need one disjunct in $\mathcal{T}$ for every answer set of Π. Thus, the size guarantee for the MODS representation of a program is rather weak, if it has many answer sets.

Corollary 136 (Size of MODS)**.** *Let Π be an answer set program. The smallest MODS representation of Π has size $\Theta(|\mathcal{AS}(\Pi)| \cdot |\mathcal{A}(\Pi)|)$.*

Example 46. *Consider the CNF*

$$C = a \vee b \vee c \wedge \neg c \vee d.$$

It can be represented in MODS as follows:

$$\begin{aligned} C \equiv \quad & a \wedge b \wedge c \wedge d \\ \vee \quad & a \wedge b \wedge \neg c \wedge d \\ \vee \quad & a \wedge b \wedge \neg c \wedge \neg d \\ \vee \quad & a \wedge \neg b \wedge c \wedge d \\ \vee \quad & a \wedge \neg b \wedge \neg c \wedge d \\ \vee \quad & a \wedge \neg b \wedge \neg c \wedge \neg d \\ \vee \quad & \neg a \wedge b \wedge c \wedge d \\ \vee \quad & \neg a \wedge b \wedge \neg c \wedge d \\ \vee \quad & \neg a \wedge b \wedge \neg c \wedge \neg d \\ \vee \quad & \neg a \wedge \neg b \wedge c \wedge d \end{aligned}$$

There are also other more intricate tractable circuit representation, which may be exponentially smaller and more importantly allow for size guarantees with respect to structural parameters such as treewidth.

One of the most prominent tractable circuit representations is sd-DNNF. It is a special case of negation normal form (NNF) [Dar04]. The latter is a rooted directed acyclic

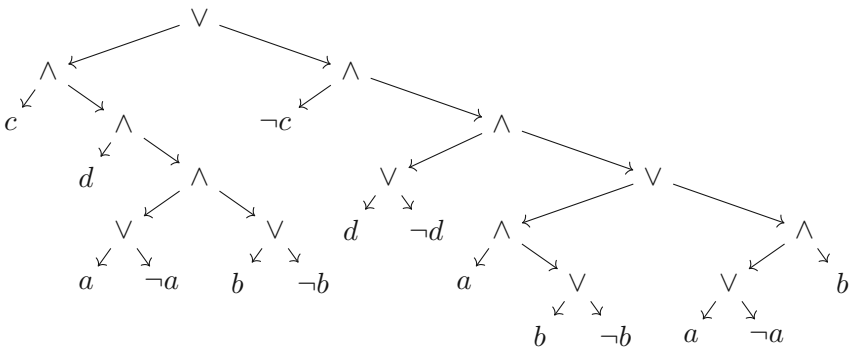


Figure 4.4: An sd-DNNFs for $C = a \vee b \vee c \wedge \neg c \vee d$.

graph in which each leaf node is labeled with either a literal, "true", or "false", and each internal node is labeled with a conjunction $\wedge$ or disjunction $\vee$. For any node n in an NNF graph, $\text{Vars}(n)$ denotes all variables in the subgraph rooted at n. In abuse of notation, we refer by n also to the formula represented by the graph n. sd-DNNF in NNF that satisfies the following additional properties (D), (d), and (s):

Decomposability (D): $\text{Vars}(n_i) \cap \text{Vars}(n_j) = \emptyset$ for any two children n_i and n_j of an and-node.

Determinism (d): $n_i \wedge n_j$ is logically inconsistent for any distinct children n_i and n_j of an or-node.

Smoothness (s): $\text{Vars}(n_i) = \text{Vars}(n_j)$ for any two children n_i and n_j of an or-node.

Example 47 (cont.)**.** *The NNF in Figure 4.4 is an sd-DNNF and equivalent to C. We observe that for each and-node, labeled $\wedge$ in the figure, the sets of variables that occur for the different children are disjoint. The NNFs of the form $a \vee \neg a$ are necessary for the circuit to be smooth. Notably, in this sd-DNNF the NNF $a \vee \neg a$ occurs twice. This is not necessary as NNFs can be DAGs. Thus, we could also point two arrows to the same NNF $a \vee \neg a$.*

A big selling point of sd-DNNFs is the fixed-parameter-tractability (FPT) result that gives a guarantee on the size and time needed to construct it.

Theorem 137 (Size of sd-DNNF [OD14])**.** *Let C be a CNF and (T, χ) a tree decomposition of $PRIM(C)$, the primal graph of C, with width k. Then there is an equivalent sd-DNNF of size $\mathcal{O}(2^k|C|)$, which can be constructed in time $\mathcal{O}(2^k poly(|C|))$ from C and (T, χ).*

This theorem even holds for a more restricted class of sd-DNNFs, namely Sentential Decision Diagrams (SDD).

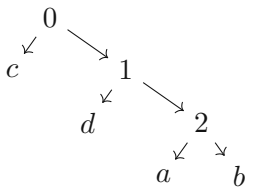


Figure 4.5: A (right-linear) vtree for the SDD in Figure 4.4.

Definition 138 (Vtree, SDD [Dar11])**.** *Let $\mathcal{V}$ be a finite set of propositional variables. A* vtree *for $\mathcal{V}$ is a rooted binary tree $T = (V, E, t_{root})$, whose leaves are labeled by variables from $\mathcal{V}$ in a one-to-one manner. Given $t \in V$, we denote by $Vars(t)$ the set of labels of leafs that occur in T below t.*

An SDD *is a d-DNNF C such that every and-node of C has exactly two children, and there exists a vtree T such that for each and-node $n = n_1 \wedge n_2$ there exists a vertex t in T with children t_1 and t_2 such that $Vars(n_i) \subseteq Vars(t_i), i = 1, 2$.*

Example 48 (cont.)**.** *The sd-DNNF is in fact also an SDD for the CNF C. This is witnessed by the vtree displayed in Figure 4.5.*

To the best of our knowledge, it is unknown whether for each sd-DNNF C an equivalent SDD C' of size polynomial in $|C|$ exists, meaning that it may be possible that sd-DNNFs can be much smaller than SDDs for some propositional theories. However, their structuredness gives us other additional possibilities. Given two SDDs C_1, C_2 that are structured by the same vtree T, there is an operator APPLY, which computes an SDD C_3 that is equivalent to $C_1 \wedge C_2$ (or $C_1 \vee C_2$) and structured by T in polynomial time in the size of C_1 and C_2 [Dar11]. This allows us to compile an SDD for a Boolean circuit without introducing auxiliary variables for the and-nodes and or-nodes (as it usually happens, when we apply the Tseitin Transformation to the circuit to obtain a CNF). This strategy allows for a different size guarantee.

Theorem 139 (Size of SDD [BS17])**.** *Let C be a Boolean circuit and (T, χ) be a tree decomposition of C with width k. Then there is an equivalent SDD of size*

$$\mathcal{O}\left(2^{2^{(k+2)2^{k+1}+1}+1} \cdot |\mathit{Vars}(C)|\right).$$

Importantly, the size of the circuit here is linear in the number of input variables of the Boolean circuit C, rather than the size of the CNF of Theorem 137. On the other hand, the dependency on the treewidth k is triple exponential, rather than single exponential.

It is not clear whether this is necessary. In fact, Amarilli, Capelli, Monet, and Senellart [Ama+20] were able to obtain a size upper bound of $\mathcal{O}\left(2^{4(k+1)} \cdot |\mathrm{Vars}(C)|\right)$ that is constructive, with time bound $\mathcal{O}\left(2^{5k} \cdot |\mathrm{Vars}(C)|\right)$, by dropping the requirements that vtrees are binary and that and-nodes should only have two children.

Another similar class that also features an APPLY operator is that of Binary Decision Diagrams (BDD). They can be defined as a special case of SDDs[2].

Definition 140 (BDD [Ake78])**.** *A* BDD *is an SDD that is structured by a right-linear vtree T, meaning that for every non-leaf node of T it holds that its left child is a variable.*

Example 49 (cont.)**.** *Since the vtree in Figure 4.5 is right-linear, the SDD in Figure 4.4 is also a BDD.*

Intuitively, the restriction on the vtree to be right-linear takes away our possibility to decompose problems of the form $C = C_1 \wedge C_2$, where C_1 and C_2 are CNFs that do not share variables, into the two subproblems C_1 and C_2. For SDDs and sd-DNNFs we can solve them independently and obtain a solution for the whole CNF C. Accordingly, BDDs come with a weaker FPT result than SDDs and sd-DNNFs.

Theorem 141 (Size of BDDs I [FPV05])**.** *Let C be a CNF and (T, χ) be a path decomposition of $PRIM(C)$, the primal graph of C, with width k. Then there is an equivalent BDD of size $\mathcal{O}(2^k|C|)$, which can be constructed in time $\mathcal{O}(2^k poly(|C|))$ from C and (T, χ).*

Note that here the guarantee is only given in terms of pathwidth instead of treewidth, which is a weaker parameter.

As with SDDs, we have a separate result for the compilation of Boolean circuits of bounded pathwidth.

Theorem 142 (Size of BDDs II [JS12; Ama+20])**.** *Let C be a Boolean circuit and (T, χ) be a path decomposition of C with width k. Then there is an equivalent BDD of size $\mathcal{O}(2^{(k+2)\cdot 2^{k+2}} \cdot |Vars(C)|)$, which can be constructed in time $\mathcal{O}(2^{(k+2)\cdot 2^{k+2}} \cdot poly(|C|))$ from C and (T, χ).*

Here in contrast to the same result for SDDs the dependency on the parameter k is double exponential rather than triple exponential. Arguably, pathwidth is a weaker parameter than treewidth, however only by at most a logarithmic factor in the number of vertices of the given graph [KS93]. In addition, since every BDD can be interpreted as an SDD, we also have the same guarantee for SDDs. Again, to the best of our knowledge it is not known whether the double exponential bound is tight [Ama+20].

This short summary of different circuit classes and known results that guarantee efficient compilation can serve us as a basis for judging different knowledge compilation based approaches to AASC. Of course, we still need to keep in mind that these theoretical guarantees may not be optimal, especially not on every instance, and that the Big-O notation hides constant factors. Furthermore, the practical efficiency also depends on how well engineered the used knowledge compilers are. Apart from this, these theoretical

[2]Strictly speaking these BDDs are known as Ordered BDDs (OBDDs)

results are only used by some knowledge compilers practically, meaning that we may even end up with worse performance. Thus, these results should not be taken as the sole basis for choosing a knowledge compilation strategy but can at least serve as an indicator from the theoretical perspective.

4.4.3 Different Approaches to the Knowledge Compilation Step

In this section, we compare the different approaches to the knowledge compilation step in the overall pipeline of AASC based on the theoretical guarantees from above. Note that this is not a complete discussion in the sense that every possible approach is discussed. Instead, we only discuss approaches for which an implementation exists. Nevertheless, we still include approaches even when their current implementation does not make use of the FPT-guarantees explicitly.

Program to MODS This approach is used by LP^{MLN} [LY17] and plingo [Hah+22]. By Corollary 136, we know that this means we need to consider each model once. Of course, finding models of a program is NP-hard but in practice this does not seem to lead to problems. When enumerating the answer sets as it is the case in LP^{MLN} and plingo, the size guarantee is used explicitly. In fact, it is not even necessary to keep all the answer sets but it is sufficient to extract a value from each answer set and then discard it. Thus, this strategy is promising, when there are "few" answer sets[3] but becomes infeasible quickly, since the linear dependency on the number of answer sets cannot be avoided.

Dynamic Programming This approach is used by dynasp [Fic+17], which is currently specialized on counting the number of answer sets rather than probabilistic reasoning or even general AASC. The following result is the basis for the implementation.

Theorem 143 ([Fic+17])**.** *Let* Π *be a ground program and let* (T, χ) *be a tree decomposition of the primal dependency graph of* Π *with width* k*. Then it is possible, assuming constant time semiring additions and multiplications, to perform AASC over* Π *in time* $\mathcal{O}(2^{2^{k+1}}|T|)$*.*

Since dynasp is based on dynamic programming over the tree decomposition, the guarantee is used explicitly.

CNF to sd-DNNF This approach has been widely considered and is for example available in ProbLog [MJ10; Fie+11]. Here, we can use the guarantees of Theorem 137, which are both in terms of size as well as time and have the lowest (i.e. single exponential) dependency on the treewidth of the primal graph of the input formula. In fact, in the context of CNFs the result is not only of theoretical interest, quite on the contrary, recent

[3] Millions of answer sets still seem to be easily feasible.

results showed that using them as a primary means of guiding the variable selection during model counting is also highly beneficial in practice [KJ21].

However, this assumes that we are given a CNF of low treewidth. A priori it is unclear whether a program of low treewidth also leads to a CNF suitable for compilation of low treewidth. We will discuss this in depth in Sections 4.5 and 4.6.

Ground Acyclic Program to SDD This is the current standard approach in the implementation of ProbLog [Fie+15]. Here, we do not first translate the program into a CNF via Clark's Completion but interpret the program as a Boolean circuit and obtain an SDD that represents the truth of the extensional atoms in terms of only the input variables. In the case of probabilistic inference, these are the probabilistic facts. Here, we obtain a triple exponential upper bound in terms of treewidth on the size of the smallest SDD via Theorem 139 or a double exponential upper bound in terms of the path width via Theorem 142. These upper bounds are not explicitly used in the implementation of ProbLog, which employs the APPLY operator on SDDs to build the SDDs in a bottom up manner, along the Boolean circuit. In order to keep the SDDs small, dynamic reordering of the vtree can be applied heuristically.

Recall that Theorem 137 also holds for compiling CNFs to SDDs [Dar11]. Thus, we could also make use of its guarantees, given that we can obtain a CNF of small treewidth from a Boolean circuit of small treewidth. On the one hand, this would lead to a single exponential instead of a triple exponential dependency on the treewidth for the worst case guarantee. On the other hand, it would also come with a higher remaining factor that uses the size of the CNF instead of the number of input variables.

Ground Program to SDD This approach is called $T_{\mathcal{P}}$-compilation [Vla+16] and is also available in the implementation of ProbLog. Since the program is not necessarily acyclic we cannot see it as a Boolean circuit. Thus, we cannot use Theorem 139 directly to obtain guarantees. However, the theorem in fact also holds when there *exists* a circuit that defines the same Boolean function as the program [BS17]. Thus, we have the same theoretical guarantees here as for the acyclic case. Practically, these guarantees are not used either since the APPLY operator is employed for compilation in a similar manner as before. Here, we need to make multiple passes over the program structure, which intuitively can be seen as a cyclic version of a Boolean circuit, until convergence of the SDDs.

Program to BDD This approach is used by the PITA system [RS11]. A theoretical guarantee can again be derived by Theorem 142. While the program is again not guaranteed to be acyclic, the theorem holds for any Boolean circuit that defines the Boolean function specified by the program [FPV05]. Thus, the worst case guarantee here is equal or worse to the one in the previous case, since we only compile to BDD and not to SDD, which encompasses BDD as a special case. As with ProbLog, the guarantees are not used explicitly, since BDDs are built in a bottom-up manner by empolying the

apply operator for BDDs. The BDDs can be dynamically minimized during compilation by heuristically adapting the variable ordering on the fly.

Ground Program to sd-DNNF Last but not least it is important to mention the strategy of Aziz, Chu, Muise, and Stuckey [Azi+15]. They modified the knowledge compiler Dsharp to work on inputs that correspond to normal answer set programs. In their approach, they did not take care of acyclicity before but during compilation by adding so called *loop formulas* [LZ04], which are clauses that prohibit cyclic derivations. This approach seems promising, since it does not need to blow up the size of the encoding with all potentially necessary acyclicity constraints but adds them only when needed. There is a downside to it however. Loop formulas as introduced by Lin and Zhao [LZ04] may span a whole SCC of the dependency graph of the program.[4] Thus, the primal graph of the CNF for Clark's Completion and all loop formulas of a program Π contains a clique for each SCC of DEP(Π). Since an SCC can potentially span large parts of the program, this may increase the treewidth drastically compared to the treewidth of the original program. Apart from that, not only the treewidth may suffer: we may end up with exponentially many additional clauses in the size of the largest SCC [LR06]. This version of Dsharp was removed from the ProbLog implementation, presumably for the above reasons.

Summary Summing up, we see that there is a wide range of different approaches to Knowledge Compilation or more generally solving AASC problems, each coming with different FPT guarantees that are only partially explicitly used. Mostly, the guarantees are exponential in a structural parameter associated with the formula that is to be compiled. The only exception is the approach of the solvers LP^{MLN} [LY17] and plingo [Hah+22] that enumerate all the models.

The most promising theoretical results are known for the compilation of CNFs to sd-DNNFs. Furthermore, the recent work of Korhonen and Järvisalo [KJ21] showed that one can achieve performance improvements by using the latter explicitly. This begs the question whether and how we can convert programs of low treewidth into CNFs of low treewidth, in order to fruitfully exploit these recent advancements.

4.5 Clark's Completion

A crucial step in the translation of programs to CNF formulas is the well-known Clark's Completion [Fag94]. While it is defined on general programs, it is only guaranteed to lead to an equivalent CNF, when the program it was applied to does not have cycles in its positive dependence graph. The traditional, most well-known version of Clark's Completion does not allow for bounds on the treewidth of the resulting CNF in terms of the treewidth of the primal graph of the original program. There has, however, already

[4] They usually contain even more variables, however already this is devastating for Knowledge Compilation.

been work by Hecher [Hec22] that does imply such bounds. In the following, we first discuss the original version of Clark's Completion and its limitations as well as the modifications by Hecher [Hec22]. After that we show that even Hecher's [Hec22] version allows for further improvements and introduce an extended version for which we can obtain better bounds.

The idea behind Clark's Completion is the following: if we have an atom a that is in the head of the rules $r_1, \ldots, r_n$, then a is only included in a stable model if we are forced to derive it due to one of these rules. More formally, the definition is as follows:

Definition 144 (Clark's Completion)**.** *Let Π be a normal answer set program. Then Clark's Completion of Π is defined as the propositional formula*

$$Clark_{Prop}(\Pi) = \bigwedge_{a \in \mathcal{A}(\Pi)} a \leftrightarrow \bigvee_{r \in \Pi_a} \bigwedge_{l \in B(r)} l,$$

where for an atom $a \in \mathcal{A}(\Pi)$, we let $\Pi_a = \{r \in \Pi \mid H(r) = a\}$.

For convenience reasons, we also introduce Clark(Π) as the CNF

$$\bigwedge_{a \in \mathcal{A}(\Pi)} \left(\neg a \vee \bigvee_{r \in \Pi_a} forced_r \wedge \bigwedge_{r \in \Pi_a} a \vee \neg forced_r \right) \wedge \bigwedge_{r \in \Pi} \left(forced_r \vee \bigvee_{l \in B(r)} \neg l \wedge \bigwedge_{l \in B(r)} \neg forced_r \vee l \right),$$

where $forced_r$ for $r \in \Pi$ is an auxiliary variable such that intuitively $forced_r$ is true iff every literal in the body of r is satisfied, which is ensured by the second line.

Formally, we obtain:

Lemma 145. *Let Π be an answer set program. Then (i) every model $\mathcal{I}$ of $Clark_{Prop}(\Pi)$ can be uniquely extended to a model $\mathcal{I}'$ of Clark(Π) by assigning $forced_r$ to true, if $\mathcal{I}' \models B(r)$ and to false, otherwise, and (ii) for every model $\mathcal{I}$ of Clark(Π) the interpretation $\mathcal{I}' = \mathcal{I} \cap \mathcal{A}(\Pi)$ is a model of $Clark_{Prop}(\Pi)$.*

The way we use Clark's Completion is to translate programs into equivalent propositional formulas. The following gives a sufficient condition for this to work.

Theorem 146 ([Fag94])**.** *Let Π be a normal answer set program. If the dependency graph $DEP(\Pi)$ of Π is acyclic, then every model of $Clark_{Prop}(\Pi)$ is an answer set of Π and vice versa.*

4.5.1 Primal Tree Decomposition Guidance

For now, we disregard the acyclicity requirement of Clark's Completion and consider only the change in treewidth as a result of applying Clark's Completion. Unfortunately, as the following example shows, we cannot give treewidth guarantees for Clark(Π) in terms of the treewidth of Π.

Example 50 (Treewidth Bounds I)**.** *Consider the program* Π_n *consisting of rules*

$$r_1 : \ a \leftarrow b_1, \ldots, r_n : \ a \leftarrow b_n$$

The treewidth of $PRIM(\Pi_n)$ *is* 1*, since the only edges are for the form* (a, b_i)*, which means that* $PRIM(\Pi_n)$ *is a tree. On the other hand, the CNF version of Clark's Completion of* Π_n *contains the following clauses to model the truth of the atom* a*:*

$$\neg a \vee \bigvee_{j=1}^{n} forced_{r_j}, \qquad a \vee \neg forced_{r_i}, \qquad \text{for } i = 1, \ldots, n$$
$$forced_{r_i} \vee \neg b_i, \qquad \neg forced_{r_i} \vee b_i, \qquad \text{for } i = 1, \ldots, n.$$

Due to the clause $\neg a \vee \bigvee_{i=1}^{n} forced_{r_i}$*, the primal graph of* $Clark(\Pi_n)$ *contains a clique of size* $n + 1$*. Therefore, the treewidth of* $Clark(\Pi_n)$ *is* n*.*

This is a problem, if we want to use the treewidth of the CNF we obtain from Clark's Completion to bound the time needed to solve an AASC instance. However, this problem can be avoided by introducing further auxiliary variables that intuitively save partial results and thus limit the dependence of the variables among each other.

Example 51 (cont.)**.** *Consider instead of the set of clauses from Example 50 the following alternative clauses:*

$$\neg a \vee part_n, \qquad a \vee \neg part_n \tag{4.6}$$
$$\neg part_i \vee b_i \vee part_{i-1}, \qquad part_i \vee \neg b_i, \qquad part_i \vee \neg part_{i-1} \qquad \text{for } i = 1, \ldots, n \tag{4.7}$$
$$\neg part_0. \tag{4.8}$$

Here, $part_i$ *for* $i = 0, 1, \ldots, n$ *are auxiliary variables. Intuitively,* $part_i$ *is responsible for storing a partial result. Namely,* $part_i$ *is true iff one of* $b_1, \ldots, b_i$ *is true and consequently there is a reason to derive* a *from one of the first* i *rules. This is ensured by the clauses in (4.8) and (4.7). Indeed, as* $part_0$ *is false, so* $part_1 \leftrightarrow b_1$ *holds as well as* $part_i \leftrightarrow part_{i-1} \vee b_i$ *for* $i = 1, \ldots, n$*. Finally, the clauses in (4.6) ensure that* a *is true iff any of the (first* n*) rules fires by ensuring that it* a *is equivalent to* $part_n$*.*

While the primal graph of this CNF has treewidth 2 *which is higher than the one of* $PRIM(\Pi_n)$ *that has treewidth* 1*, it is independent of* n*, in contrast to the treewidth of* $Clark(\Pi_n)$*.*

The idea of the previous example is formalized as $\mathrm{PClark}(\Pi, \mathcal{T}, t_r)$ a revised definition of $\mathrm{Clark}_{\mathrm{Prop}}(\Pi)$ given by Hecher [Hec22] that uses a tree decomposition $\mathcal{T}$ with root t_r of the primal graph of Π for guiding the Clark Completion. It leads to the following guarantee:

Theorem 147 ([Hec22])**.** *Given a normal answer set program Π and a tree decomposition $\mathcal{T} = (T, \chi)$ of $PRIM(\Pi)$ with width k and root t_r, the CNF $PClark(\Pi, \mathcal{T}, t_r)$ can be constructed in time linear in $|\Pi| + |T|$ and satisfies that*

(i) *every model of $Clark_{Prop}(\Pi)$ can be uniquely extended to a model of $PClark(\Pi, \mathcal{T}, t_r)$,*

(ii) *the size of $PClark(\Pi, \mathcal{T})$ is in $\mathcal{O}(k|\Pi|)$, and*

(iii) *the treewidth of $PRIM(PClark(\Pi, \mathcal{T}, t_r))$ is at most $3(k+1)$.*

Intuitively, we can avoid a higher increase in treewidth by avoiding long clauses introduced by large disjunctions $a_1 \vee \cdots \vee a_n$. For this, we apply the Tseitin transformation to obtain multiple shorter clauses that correspond to $a_1 \vee a_2 \vee \mathit{part}_1, \mathit{part}_1 \leftrightarrow a_3 \vee \mathit{part}_2, \ldots, \mathit{part}_{n-3} \leftrightarrow a_{n-1} \vee a_n$.

We do not spell out PClark(.), as we will give a further improved version. Notably, Clark's Completion may also introduce large conjunctions, which we also wish to avoid if they artificially increase the treewidth.

4.5.2 Incidence Tree Decomposition Guidance

We have seen that we can use a tree decomposition of the primal graph of a program to limit the increase in the treewidth during translation to CNF that is caused by many rules that have the same atom in the head. Another factor that can lead to artificially high treewidth are long rules:

Example 52 (Treewidth Bounds II)**.** *Consider the program Π_n with the single rule*

$$r: \ a \leftarrow b_1, \ldots, b_n.$$

The treewidth of $PRIM(\Pi_n)$ is n, since it is a clique on $n+1$ vertices. Thus, Theorem 147 guarantees us a CNF of treewidth at most $3(n+1)$. In this case, this upper bound is not helpful, since already the usual version of Clark's Completion results in the following clauses to model the truth of atom a:

$$a \vee \neg \mathit{forced}_r, \qquad \neg a \vee \mathit{forced}_r,$$
$$\mathit{forced}_r \vee \bigvee_{i=1}^{n} \neg b_i, \qquad \bigwedge_{i=1}^{n} (\neg \mathit{forced}_r \vee b_i)\,.$$

The primal graph of this CNF also has treewidth n, which is better than our upper bound.

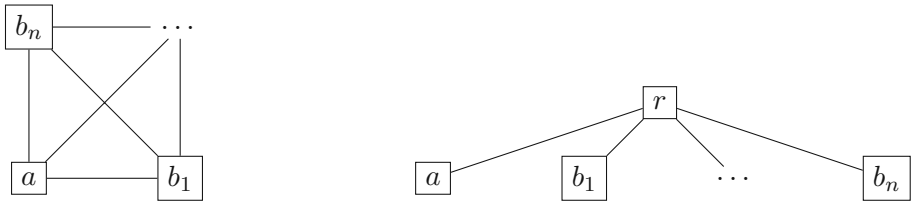


Figure 4.6: The primal graph (left) and the incidence graph of the program Π_n from Example 53.

We can avoid the long clauses by using the following propositional formula instead:

$$\begin{aligned} &a \leftrightarrow b_1 \wedge part_1, \\ &part_i \leftrightarrow b_{i+1} \wedge part_{i+1}, && \text{for } i = 1, \ldots, n-3, \\ &part_{n-2} \leftrightarrow b_{n-1} \wedge b_n. \end{aligned}$$

The treewidth of the primal graph of the corresponding CNF is 2 *and therefore independent of* n*.*

As previously, we see that we can decrease the treewidth of the CNF resulting from Clark's Completion by introducing auxiliary variables. The difference here is that we introduce them for conjunctions instead of disjunctions. Unfortunately, as we have also seen in the example, a tree decomposition of the primal graph of the program is unlikely to help us: it also assigns a high treewidth to the program even if it has a simple representation.

We overcome this problem by considering the incidence graph of the program instead of the primal graph.

Example 53 (cont.)**.** *The incidence graph of* Π_n *in Example 52 is shown in Figure 4.6; as it is a tree, it has treewidth* 1*.*

Definition 148 (Incidence Tree Decomposition-guided Clark's Completion)**.** *Let* Π *be a normal answer set program and* $\mathcal{T} = (T, \chi)$ *a tree decomposition of* $INC(\Pi)$ *with root* t_r*. Then* $IClark(\Pi, \mathcal{T}, t_r)$ *the* Incidence Tree Decomposition-guided *Clark's Completion of* Π *with respect to* (T, χ) *and root* t_r *is defined as the propositional formula*

$$upto_t^a \leftrightarrow \bigvee_{t' \in children(t), a \in \chi(t')} upto_{t'}^a \vee \bigvee_{r \in \chi(t), H(r) = a} forced_r \qquad (4.9)$$

$$\text{for } t \in T, a \in \chi(t)$$

$$a \leftrightarrow upto_{t'}^a \qquad (4.10)$$

$$\text{for } t, t' \in T, a \in \chi(t') \setminus \chi(t), t' \in children(t)$$

$$a \leftrightarrow upto_{t_r}^a \qquad (4.11)$$

$$\text{for } a \in \chi(t_r)$$

$$upto_t^r \leftrightarrow \bigwedge_{t' \in children(t), r \in \chi(t')} upto_{t'}^r \wedge \bigwedge_{b \in \chi(t), b \in B(r)} b \tag{4.12}$$

$$\text{for } t \in T, r \in \chi(t)$$

$$forced_r \leftrightarrow upto_{t'}^r \tag{4.13}$$

$$\text{for } t, t' \in T, r \in \chi(t') \setminus \chi(t), t' \in children(t)$$

$$forced_r \leftrightarrow upto_{t_r}^r \tag{4.14}$$

$$\text{for } r \in \chi(t_r)$$

The basic intuition here is the same as with the original CNF version of Clark's Completion, i.e., we introduce additional variables that are defined as equivalent to subformulas. The variables of the form $upto_t^a$ for $t \in T$ and $a \in \mathcal{A}(\Pi)$ intuitively capture whether a is derived by any of the rules that occur in $\chi(t)$ or $\chi(t')$ for a descendant t' of t. This is ensured in (4.9). Then in (4.10) and (4.11) we ensure that the atom a holds if it was derived by any rule, by checking whether it was derived in the bag of the tree decomposition that contains it and is closest to the root (which may be the root). Similarly, the atoms of the form $upto_t^r$ for $t \in T$ and $r \in \Pi$ intuitively capture whether the all the body atoms of r that occur in $\chi(t)$ or $\chi(t')$ for all descendants t' of t are satisfied due to (4.12). Accordingly, in (4.13) and (4.14) we ensure that $forced_r$ holds iff all the body atoms of r are satisfied, by checking whether $upto_t^r$ holds in the unique bag $t \in T$ such that $r \in \chi(t)$ and t does not have any ancestors that contain r.

This leads to the following result as desired:

Theorem 149. *Given a normal answer set program Π and a tree decomposition $\mathcal{T} = (T, \chi)$ of $INC(\Pi)$ with width k and root t_r, the CNF $IClark(\Pi, \mathcal{T}, t_r)$ can be constructed in time linear in $|\Pi| + |T|$ and satisfies that*

(i) *every model of $Clark_{Prop}(\Pi)$ can be uniquely extended to a model of $IClark(\Pi, \mathcal{T}, t_r)$,*

(ii) *the size of $IClark(\Pi, \mathcal{T})$ is in $\mathcal{O}(k|\Pi|)$, and*

(iii) *the treewidth of $PRIM(IClark(\Pi, \mathcal{T}, t_r))$ is at most $3(k+1)$.*

Proof (Sketch, for the full proof see Appendix C.1). Roughly, we construct a new tree decomposition by adding for each node $t \in T$ with vertices $\chi(t)$ the atoms $upto_t^x$ for each $x \in \chi(t)$ and $forced_r$ for $r \in \chi(t)$. Then we are interested in the cardinality of the set

$$(\chi(t) \setminus \{r \mid r \in \chi(t) \cap \Pi\}) \cup \{forced_r \mid r \in \chi(t) \cap \Pi\} \cup \{upto_t^x \mid x \in \chi(t)\},$$

which is $2(k+1)$. However, this does not cover the clauses in (4.12) and (4.9). We can assume w.l.o.g. that every node $t \in T$ has at most 2 children (e.g. by introducing duplicates of t) and add $upto_{t'}^x$ for every child t' of t to naively achieve an upper bound of $4(k+1)$. By instead using in the new tree decomposition copies $(t,1), \ldots, (t, |\chi(t)|+1)$ of t for each $t \in T$ and processing the clauses in (4.12) and (4.9) one by one, we achieve

the upper bound of $3(k+1)$. Assuming $\chi(t) = \{x_1, \ldots, x_{|\chi(t)|}\}$, we start with the bag $(t, |\chi(t)|+1)$ that contains $upto_t^{x_{|\chi(t)|}}, x, upto_{t_1}^x$, and $upto_{t_2}^x$ for the children t_1, t_2 of t and all $x \in \chi(t)$. Then, we define $\chi'(t,i)$ for $i = 1, \ldots, |\chi(t)|$ as

$$\{upto_t^{x_{i-1}}\} \cup \chi'(t, i+1) \setminus \{upto_t^{x_i}, upto_{t_1}^{x_i}, upto_{t_2}^{x_i}\}.$$

This means that we start with bag $(t, |\chi(t)|+1)$ of size $3(k+1)+1$ and decrease the size of the following bags by 2 in each step, resulting in a width of at most $3(k+1)$. □

Recall that a graph of primal treewidth k has incidence treewidth less or equal to $k+1$. Thus, in the worst case the guarantees of Theorem 149 only differ from those of Theorem 147 by a constant factor. However, since incidence treewidth may be arbitrarily smaller than primal treewidth, the guarantees with respect to incidence treewidth can be much better. The latter can also be seen in Example 53, where we consider a family of programs with constant incidence treewidth but unbounded primal treewidth. Thus, Theorem 149 broadens the range of cases, where we can fruitfully use CNF-translation followed by algebraic model counting on the CNF for algebraic answer set counting.

In order to solve AASC with knowledge compilation via a version of Clark's Completion, we however first need to ensure that the resulting CNF preserves the models of the original program bijectively.

4.6 Cycle Breaking

Recall that Clark's Completion is only guaranteed to capture the answer sets of a program if the program is acyclic. Thus, for a program with a cyclic dependency graph, we first need to make some further effort to ensure that Clark's Completion will be correct. This process is commonly referred to as cycle breaking and transforms a program Π into a program $\mathcal{C}(\Pi)$.

There has been a lot of work on cycle breaking both in the ASP community [LZ03; Jan04; Hec22] for translations to SAT as well in the probabilistic reasoning community [MJ10] specifically for weighted model counting. The basic idea of cycle breaking is to modify the program in such a way that afterwards the models of the Clark Completion and the modified program are the same. In the context of general AASC, it is moreover important that the answer sets before and after cycle breaking are in a specific one-to-one relationship. Formally, the cycle breaking needs to be faithful:

Definition 150 (Faithfulness)**.** *A cycle breaking $\mathcal{C}(.)$ is* faithful *(for Π), if:*

(i) $|\mathcal{AS}(\Pi)| = |\mathcal{AS}(\mathcal{C}(\Pi))|$, *and*

(ii) $\mathcal{AS}(\Pi) = \{\mathcal{I} \cap \mathcal{A}(\Pi) \mid \mathcal{I} \in \mathcal{AS}(\mathcal{C}(\Pi))\}$.

As expected, faithfulness guarantees that we can perform AASC over the program obtained by cycle breaking without changing the result.

Lemma 151 (Faithfulness Implies Query Invariance)**.** *Let $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$ be a measure and let $\mathcal{C}(.)$ be a faithful cycle breaking for Π. Then for $\mu' = \langle \mathcal{C}(\Pi), \alpha, \mathcal{R} \rangle$ it holds that $\mu(a) = \mu'(a)$ for every $a \in \mathcal{A}(\Pi)$.*

The proof of this lemma can be found in Appendix C.2.

The translations of Lin and Zhao [LZ03] and Hecher [Hec22] do not satisfy this requirement for all program, as they serve to check the existence of an answer set.

In contrast, the two cycle breakings of Janhunen and Niemelä [JN11] and Mantadelis and Janssens [MJ10] are faithful for all programs. In fact, both have been considered in the context of probabilistic reasoning. Janhunen and Niemelä's [JN11] was considered for the implementation of Problog (cf. [Fie+11]) and Mantadelis and Janssens' [MJ10] is still part of the standard Problog implementation.

In this section, our aim is to find a cycle breaking that will result in good AASC-performance. As mentioned before, our main criterion here is whether we can bound the increase of the treewidth. However, of course also the increase in program size and insights regarding the semantic complexity are relevant for us. In the following, we thus first analyze the state of the art of faithful cycle breakings. Afterwards we introduce a novel cycle breaking called T$_\mathcal{P}$-Unfolding, which we use in our implementation, and argue that it has favorable properties.

4.6.1 Necessity of Cycle Breaking

Before we consider existing cycle breakings, we first motivate the need for it by giving an example program, where the models of Clark's Completion do not align with the stable models.

Example 54. *Consider again the program Π_{sm} from the running example.*

$$
\begin{array}{rrr}
\{st(1)\} \leftarrow & \{st(2)\} \leftarrow & \{st(3)\} \leftarrow \\
\{inf(3,1)\} \leftarrow & \{inf(1,2)\} \leftarrow & \{inf(2,3)\} \leftarrow \\
sm(1) \leftarrow st(1) & sm(1) \leftarrow inf(3,1), sm(3) & \\
sm(2) \leftarrow st(2) & sm(2) \leftarrow inf(1,2), sm(1) & \\
sm(3) \leftarrow st(3) & sm(3) \leftarrow inf(2,3), sm(2) &
\end{array}
$$

Recall that $\{a\} \leftarrow$ is short for the two rules $a \leftarrow not\, na$ and $na \leftarrow not\, a$. Thus, each of the stress and influence atoms, contribute the following equivalences to Clark's Completion, respectively:

$$
\begin{array}{rr}
st(i) \leftrightarrow \neg nst(i) & ninf(i,j) \leftrightarrow \neg inf(i,j) \\
nst(i) \leftrightarrow \neg st(i) & inf(i,j) \leftrightarrow \neg ninf(i,j)
\end{array}
$$

Without any additional constraints, this means that we can choose the truth values of $st(i)$ and $inf(i,j)$ arbitrarily as long we assign $nst(i)$ and $ninf(i,j)$ the negation, respectively.

Since this is the intended behavior of choice constraints, we see that Clark's Completion works as expected here. On the other hand, for the smokes atoms, we obtain the following equivalences

$$sm(i) \leftrightarrow st(i) \vee (inf(j,i) \wedge sm(j)),$$

where $i + 1 \equiv j \mod 3$. *I.e. i smokes iff i is stressed or j influences i and j smokes. Under the stable model semantics, we can only derive that any person smokes if at least one person is stressed. However, for Clark's Completion there is a model, where every person smokes but nobody is stressed, namely*

$$\{sm(1), sm(2), sm(3), nst(1), nst(2), nst(3), inf(3,1), inf(1,2), inf(2,3)\}.$$

While this is also a model of Π_{sm} it is not stable, since the strict subset

$$\{nst(1), nst(2), nst(3), inf(3,1), inf(1,2), inf(2,3)\}$$

is also a model of Π_{sm}.

The previous example clearly shows that the problem with Clark's Completion on general problem is caused by cyclic derivations. Cyclic derivations are those that use that an atom a is true to derive that a is true.

4.6.2 The MJ(.) Cycle Breaking [MJ10]

The strategy of Mantadelis and Janssens' [MJ10] cycle breaking MJ(.) to avoid such cyclic derivations is to introduce copies a_F of an atom a for $F \subseteq \mathcal{A}(\Pi)$ that intuitively capture derivations of a that do not positively use any atom from F. Then we can use the atom $a_{\{b\}}$ in a derivation of b from a, even if a and b are in a cyclic dependency. Naturally, introducing a copy a_F for every subset $F \subseteq \mathcal{A}(\Pi)$ is undesirable as it would lead to an exponential number of atoms.

In order to avoid this whenever possible, the cycle breaking MJ(.) makes use of two ideas. Firstly, it uses the fact that sometimes the derivations for a that do not use atoms from F or F' align. Then it does not make sense to introduce two separate atoms $a_F, a_{F'}$ and one can use one of them for both cases. Second, if every derivation for b uses the atoms in $F \subseteq \mathcal{A}(\Pi)$ before using the atom a, then no atom $a_{\{b\}}$ will be created but only an atom $a_{F \cup \{b\}}$.

We illustrate the idea behind MJ(.) on our running example.

Example 55 (cont.)**.** *First consider the rules for the guesses of the stressed and influences predicate.*

$$\begin{array}{lll} \{st(1)\} \leftarrow & \{st(2)\} \leftarrow & \{st(3)\} \leftarrow \\ \{inf(3,1)\} \leftarrow & \{inf(1,2)\} \leftarrow & \{inf(2,3)\} \leftarrow \end{array}$$

Here, we do not need to change anything, since these atoms are not involved in any cycles. Next, we consider the remaining atoms in the rules

$$sm(1) \leftarrow st(1) \qquad sm(1) \leftarrow inf(3,1), sm(3)$$
$$sm(2) \leftarrow st(2) \qquad sm(2) \leftarrow inf(1,2), sm(1)$$
$$sm(3) \leftarrow st(3) \qquad sm(3) \leftarrow inf(2,3), sm(2)$$

Here, $sm(1), sm(2)$ *and* $sm(3)$ *are in a cyclic dependency. We first consider* $sm(1)$*. Due to the cycle, we cannot use all derivations of* $sm(3)$ *to derive* $sm(1)$ *using the rule*

$$sm(1) \leftarrow inf(3,1), sm(3).$$

Instead, we introduce a copy $sm(3)_{\{sm(1)\}}$ *of the atom* $sm(3)$ *that captures all derivations of* $sm(3)$ *that do not make use of* $sm(1)$*:*

$$sm(1) \leftarrow st(1) \qquad sm(1) \leftarrow inf(3,1), sm(3)_{\{sm(1)\}}$$

Similarly, in order to capture all derivations of $sm(3)$ *from* $inf(2,3), sm(2)$ *using the rule*

$$sm(3) \leftarrow inf(2,3), sm(2)$$

we create a copy $sm(2)_{\{sm(1),sm(3)\}}$ *of the atom* $sm(2)$ *that captures all derivations of* $sm(2)$ *that use neither* $sm(1)$ *nor* $sm(3)$*:*

$$sm(3)_{\{sm(1)\}} \leftarrow st(3) \qquad sm(3)_{\{sm(1)\}} \leftarrow inf(2,3), sm(2)_{\{sm(1),sm(3)\}}$$

Finally, to capture the derivations of $sm(2)$ *that use neither* $sm(1)$ *nor* $sm(3)$ *we use the rule*

$$sm(2)_{\{sm(1),sm(3)\}} \leftarrow st(2).$$

Here, we cannot use the rule

$$sm(2) \leftarrow inf(1,2), sm(1)$$

since it would make use of $sm(1)$*.*

The derivations for $sm(2)$ *and* $sm(3)$ *are handled analogously to those of* $sm(1)$ *using the following rules:*

$$sm(2) \leftarrow st(2) \qquad sm(2) \leftarrow inf(1,2), sm(1)_{\{sm(2)\}}$$
$$sm(1)_{\{sm(2)\}} \leftarrow st(1) \qquad sm(1)_{\{sm(2)\}} \leftarrow inf(3,1), sm(3)_{\{sm(2),sm(1)\}}$$
$$sm(3)_{\{sm(2),sm(1)\}} \leftarrow st(3)$$
$$sm(3) \leftarrow st(3) \qquad sm(3) \leftarrow inf(2,3), sm(2)_{\{sm(3)\}}$$
$$sm(2)_{\{sm(3)\}} \leftarrow st(2) \qquad sm(2)_{\{sm(3)\}} \leftarrow inf(1,2), sm(1)_{\{sm(3),sm(2)\}}$$
$$sm(1)_{\{sm(3),sm(2)\}} \leftarrow st(1)$$

Note that here we only used the second idea of Mantadelis and Janssens' [MJ10] cycle breaking, e.g. by not creating an atom $sm(1)_{\{sm(3)\}}$ *since every derivation of* $sm(3)$ *that uses* $sm(1)$ *also uses* $sm(2)$*. The first idea does not apply here, since the derivations for* $sm(1)$ *without* $\{sm(2)\}$ *and without* $\{sm(3), sm(2)\}$ *differ (and similarly for the derivations of* $sm(2), sm(3)$*).*

This brings us to the question of how the treewidth changes when we apply this cycle breaking. The strategy in Theorem 149 was intuitively to add every copy of an atom to every bag in a tree decomposition that contains that atom. If we do the same with every copy a_F for an atom a and $F \subseteq \mathcal{A}(\Pi)$, we only get an upper-bound that is exponential in the number of atoms of Π. We may do slightly but not significantly better as the following two theorems show.

We first recall that a *simple cycle* in a directed graph G is a (directed) path in G from a vertex $v \in V(G)$ to v such that the only vertex that is visited twice is v.

Theorem 152. *Let* Π *be a normal answer set program of treewidth* k *such that the largest strongly connected component of* $DEP(\Pi)$ *has size* s *and any strongly connected component of* $DEP(\Pi)$ *has at most* c *directed simple cycles. Then the treewidth of* $MJ(\Pi)$ *is in* $\mathcal{O}(k \cdot s \cdot c)$*.*

As in Theorem 149, we add all copies of a to every bag that contains atom a to obtain the desired result.

Proof. Let (T, χ) be a tree decomposition of $\mathrm{PRIM}(\Pi)$. Then (T, χ'), where

$$\chi'(t) = \chi(t) \cup \{a_F \mid F \subseteq \mathcal{A}(\Pi), a_F \in \mathcal{A}(\mathrm{MJ}(\Pi))\}$$

is a tree decomposition of $\mathrm{MJ}(\Pi)$. This follows from the fact that (T, χ) is a tree decomposition of the original program and since for every rule

$$r_c = a_F \leftarrow b_{1F_1}, \ldots, b_{nF_n}, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_m$$

in $\mathrm{MJ}(\Pi)$ there is a rule

$$r = a \leftarrow b_1, \ldots, b_n, \mathrm{not}\, c_1, \ldots, \mathrm{not}\, c_m$$

in Π. From the definition of χ' it follows that $r_c \in \chi'(t)$ iff $r \in \chi(t)$. Therefore, every rule is contained completely in some bag as required. □

This upper bound is not very helpful, since the strongly connected components of $\mathrm{DEP}(\Pi)$ can be large and can have exponentially many directed simple cycles in their size. Unfortunately, we cannot do much better.

Theorem 153. *There is a family of programs* $(\Pi_n)_{n \in \mathbb{N}}$ *such that*

1. *DEP*(Π_n) *has exactly one simple cycle,*
2. *the treewidth of* Π_n *is bounded by a constant (independent of* n*),*
3. *the number of atoms and rules of* Π_n *is linear in* n*, and*
4. *the treewidth of MJ(*Π_n*) grows linearly with* n*.*

Proof (Sketch, for the full proof see Appendix C.2). We define a family of programs as follows

$$\begin{aligned}\Pi_n = &\{\{v(i)\} \leftarrow \mid i = 1, \ldots, n\} \cup \\ &\{\{e(i,j)\} \leftarrow \mid i, j = 1, \ldots, n, i+1 \equiv j \mod n\} \cup \\ &\{in(i) \leftarrow v(i) \mid i = 1, \ldots, n\} \cup \\ &\{in(i) \leftarrow e(i,j), in(j) \mid i, j = 1, \ldots, n, i+1 \equiv j \mod n\}.\end{aligned}$$

Intuitively, Π_n takes the directed graph over n vertices with arcs

$$(1,2), (2,3), \ldots, (n-1,n), (n,1),$$

thus inducing exactly one cycle $1, 2, \ldots, n, 1$. Then it guesses a random subset of its vertices ($v(i)$) and arcs ($e(i,j)$). All vertices are kept such that $v(i)$ holds or the edge of the predecessor is present $e(i,j)$ and the vertex j was kept. In other words, the latter means that all vertices on maximal paths of guessed edges, whose starting vertex was guessed are also kept.

1.-3. can be easily verified. To show 4., we show that PRIM(MJ(Π_n)) has a graph minor such that every vertex has at least degree n. From this it follows that the treewidth of PRIM(MJ(Π_n)) is at least n by employing standard results from the treewidth literature [KBvH01; Bod98]. □

We see that the cycle breaking MJ(.) does not have desirable properties when it comes to treewidth.

4.6.3 The JN(.) Cycle Breaking [JN11]

We continue with the second cycle breaking JN(.) due to Janhunen and Niemelä [JN11]. The previous strategy was to capture partial derivations explicitly by introducing auxiliary atoms leading to an acyclic and answer set preserving program. The cycle breaking JN(.) introduced in this work instead leaves the original program untouched but adds constraints that ensure each true atom can be derived. For this, it makes use of level rankings [Nie08]. Intuitively, the levels of atoms specify in which order they can be used in a derivation, i.e., when the level of atom a is lower than that of atom b, then we can use a rule that has b in the head and a in the body to derive b. It was shown that it is possible to characterize the answer sets of a program in terms of Clark's Completion in

combination with level ranking constraints. Furthermore, by adding constraints on the level ranking we obtain a one-to-one correspondence [Nie08].

We illustrate the idea behind JN(.) on our running example and refer the interested reader to [JN11] for details.

Example 56 (cont.)**.** *As already mentioned, we keep the original set of rules untouched. That is JN(*Π*)* $= \Pi \cup \Pi'$*, where* Π' *contains additional rules that specify level ranking constraints.*

The level ranking constraints for Π_{sm} *are as follows. As before, the guesses of the stressed and influences predicates do not require additional rules. More generally, atoms a that can only be derived from rules whose positive body atoms are not in a cycle with a do not require level ranking constraints.*

Thus, we only need to take care of the following rules:

$$sm(1) \leftarrow st(1) \qquad sm(1) \leftarrow inf(3,1), sm(3)$$
$$sm(2) \leftarrow st(2) \qquad sm(2) \leftarrow inf(1,2), sm(1)$$
$$sm(3) \leftarrow st(3) \qquad sm(3) \leftarrow inf(2,3), sm(2)$$

For $sm(1)$*, this leads to the following additional rules:*

$$\mathbf{just}(sm(1)) \leftarrow st(1) \tag{4.15}$$
$$\mathbf{just}(sm(1)) \leftarrow inf(3,1), sm(3), \mathbf{lt}(sm(3), sm(1)) \tag{4.16}$$
$$\leftarrow sm(1), not\,\mathbf{just}(sm(1)) \tag{4.17}$$
$$\mathbf{next}(sm(1)) \leftarrow st(1) \tag{4.18}$$
$$\mathbf{next}(sm(1)) \leftarrow inf(3,1), sm(3), \mathbf{succ}(sm(3), sm(1)) \tag{4.19}$$
$$\leftarrow sm(1), not\,\mathbf{next}(sm(1)) \tag{4.20}$$

Here, we introduce the following new atoms in order to model the level ranking constraints:

- $\mathbf{just}(sm(1))$ *means that* $sm(1)$ *is justified, i.e., there is a rule that is applicable also when taking into account level rankings.*
- $\mathbf{lt}(sm(3), sm(1))$ *means that* $sm(3)$ *is at a lower level than* $sm(1)$*. In this case* $sm(3)$ *can be used in a derivation of* $sm(1)$*.*
- $\mathbf{next}(sm(1))$ *means that* $sm(1)$ *is at the next level compared to some atom that is used to derive it. I.e., there is a rule with* $sm(1)$ *in the head and an atom in the body that is exactly one level lower than that of* $sm(1)$*. E.g., in (4.19) we derive* $\mathbf{next}(sm(1))$ *when we can derive* $sm(1)$ *from* $inf(3,1)$ *and* $sm(3)$ *and the level of* $sm(1)$ *is equal to* 1 *plus the level of* $sm(3)$*.*
- $\mathbf{succ}(sm(3), sm(1))$ *means that the level of* $sm(3)$ *is exactly one level lower than the level of* $sm(1)$*.*

The atoms of the form $\mathbf{just}(a)$ *and* $\mathbf{next}(a)$ *are defined by the given rules. For the other auxiliary atoms of the form* $\mathbf{succ}(b, a)$ *and* $\mathbf{lt}(b, a)$ *we still need to add definitions based on the levels associated with the atoms* a *and* b*. To model the level of an atom* a*, we use atoms that act as a binary counter that represents the level. I.e., if the SCC of the dependency graph of the program that contains* a *has size* n*, we introduce atoms* $\mathbf{bin}(a, i)$ *that are guessed via rules* $\{\mathbf{bin}(a, i)\} \leftarrow$ *for* $i = 0, \ldots, \lceil \log_2(n) \rceil$*. Then for a given interpretation* $\mathcal{I}$ *of these atoms the level of* a *is given by*

$$\sum_{i=0}^{\lceil \log_2(n) \rceil} \begin{cases} 2^i & \text{if } \mathbf{bin}(a, i) \in \mathcal{I}, \\ 0 & \text{otherwise.} \end{cases}$$

Using this correspondence, we can define the atoms $\mathbf{succ}(b, a)$ *and* $\mathbf{lt}(b, a)$ *in terms of the binary counter atoms of* a *and* b*. Note that in order to obtain a faithful translation, we additionally need to ensure that the level of atoms is minimal, as otherwise it is not unique. For details, we refer to the original paper [JN11].*

This brings us to the question of treewidth guarantees that we can give for JN(Π). In contrast to MJ(Π), we do not need information about the number of simple cycles for this.

Theorem 154. *Let* Π *be a normal answer set program of treewidth* k *such that the largest strongly connected component of DEP*(Π) *has size* s*. Then the treewidth of JN(*Π*) is in* $\mathcal{O}(k^2 + k \log_2(s))$*.*

As with Theorem 149 the idea here is to take an existing tree decomposition of optimal width k and add to each bag all auxiliary atoms that are related to the atoms in the bag. For the proof see Appendix C.2.

We point out that the quadratic dependency on the treewidth k can actually be avoided by modifying Janhunen and Niemelä's [JN11] definition of cycle breaking. For this, observe that the quadratic dependency stems from the inclusion of $\mathbf{lt}(b, a)$ and $\mathbf{succ}(b, a)$ for every $a, b \in \chi(t)$ in the new bag $\chi'(t)$. If we instead define one atom $\mathbf{lt}(b, a, r)$ and $\mathbf{succ}(b, a, r)$ per rule r such that $head(r) = a$ and $b \in body(r)$, we do not need to include all such atoms in the same bag but can handle them in different bags. It follows that there is a cycle breaking with a treewidth upper bound in $\mathcal{O}(k \log_2(s))$. In fact, we observed that the implementation of JN(.) in LP2LP2 does exactly this, meaning that it comes with a treewidth upper bound of $\mathcal{O}(k \log_2(s))$.

We see that JN(.) gives better guarantees than MJ(.), since $k \log_2(s)$ grows slower asymptotically than $k \cdot s \cdot c$ even when $c = 1$. However, it uses a binary counter for the encoding of the level ranking. While this provides very good asymptotic size guarantees, both in terms of the resulting program and its treewidth, binary encodings can sometimes have negative effects on the practical performance of SAT solvers [Bjö11] and can result in much slower performance than encodings of a larger asymptotic size.

4.6.4 $T_{\mathcal{P}}$-Unfolding

We have seen that the current cycle breakings either come with weak upper bounds on the treewidth or have good upper bounds but use an encoding that can lead to impairments of the practical performance of SAT solvers and therefore also of knowledge compilers.

We want to avoid both drawbacks as far as possible. For this, we take a closer look at dependency graphs of programs and introduce cbs(G) a novel parameter, the *component-boosted backdoor size* of a digraph. Intuitively, cbs(.) (defined later in Definition 160) provides a more fine grained measure of directed cyclicity. More importantly, we can exploit it in $T_{\mathcal{P}}$-Unfolding, our new cycle breaking, which leads to the following guarantees:

Theorem 155. *For any factorized measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$, we can construct in polynomial time in the size of Π given access to an NP-oracle a factorized measure $\mu' = \langle \Pi', \alpha, \mathcal{R} \rangle$ with an acyclic program Π' such that*

(i) *for all $a \in \mathcal{A}(\Pi)$ it holds that $\mu(a) = \mu'(a)$,*

(ii) *the treewidth of Π' is at most $k \cdot cbs(DEP(\Pi))$, where k is the treewidth of Π, and*

(iii) *the size of Π' is at most $cbs(DEP(\Pi)) \cdot |\Pi|$.*

Here, the increase of the treewidth and the program is bounded by cbs(DEP(Π)). Thus, this factor only depends on the dependency graph of Π and its cyclicity, rather than the number of atoms in the program or the size of the largest SCC of cbs(DEP(Π)).

As the name says, $T_{\mathcal{P}}$-Unfolding is related to another approach for AASC called $T_{\mathcal{P}}$-compilation [Vla+16]. Intuitively, the idea of $T_{\mathcal{P}}$-compilation is to capture the derivations of atoms by iteratively compiling SDDs that are increasingly precise approximations thereof. For this, we take at each step all rules $r_1, \ldots, r_m$ in Π such that $head(r_i) = a$ for some atom $a \in \mathcal{A}(\Pi)$. Then, we first conjoin the SDDs that represent the derivations of the body atoms of each rule r_i. By disjoining the resulting SDDs for each rule r_i, we get a better approximation of the derivations of a.

We use a similar idea to iteratively capture increasingly more derivations of atoms, but in a different way. Namely, we introduce at each step a new atom $a^{(i)}$ that represents the set of derivations of a that are currently captured. Intuitively, these new atoms $a^{(i)}$ are similar to the atoms a_F introduced by Mantadelis and Janssens' [MJ10] cycle breaking, since both capture a subset of the derivations of a. However, the captured subsets of derivations are different, which allows us to obtain better theoretical guarantees for $T_{\mathcal{P}}$-Unfolding.

$T_{\mathcal{P}}$-Unfolding, which takes as input a program Π and an *unfolding sequence* $s \in \mathcal{A}(\Pi)^*$, which is a list of atoms $s_1 \ldots s_n$, with $s_i \in \mathcal{A}(\Pi)$, is described in Algorithm 2. Intuitively, where the immediate consequence operator $T_{\mathcal{P}}$ [vEK76] checks if an atom a follows from previously derived atoms, $T_{\mathcal{P}}$-Unfolding introduces copies of all rules that derive a from

Algorithm 2 $T_{\mathcal{P}}$-UNFOLD(Π, s)

Input A program Π and an unfolding sequence $s \in \mathcal{A}(\Pi)^*$.
Output An acyclic program Π'.

```
1:  last = {a ↦ ⊥ | a ∈ A(Π)}
2:  cnt = {a ↦ 0 | a ∈ A(Π)}
3:  Π' = {r ∈ Π, H(r) = ⊥}
4:  for i = 1, ..., len(s) do
5:      if isLastOccurrence(s_i, i, s) then
6:          head = s_i
7:      else
8:          head = s_i^{cnt(s_i)+1}
9:      for r ∈ Π, s_i = H(r) do
10:         B+_new = {last(b) | b ∈ B+(r)}
11:         Π' = Π' ∪ {head ← B+_new, B−(r)}
12:     last(s_i) = head
13:     cnt(s_i) = cnt(s_i) + 1
14: return Π'
```

previously considered atoms. For this, we iterate over the unfolding sequence $s = s_1 \dots s_n$, considering s_i at the i^{th} step (line 4). As the head atom of the rule-copies we take a new copy $s_i^{cnt(s_i)+1}$ or the original atom s_i depending on whether s_i occurs again in $s_{i+1} \dots s_n$ (see lines 5-8). The positive body atoms are replaced by the last copy made of them (line 10) and the negative atoms in $B^-(r)$ are left as they are. After copying all rules with s_i in the head, we update the last copy of s_i and increase the counter storing the number of copies (lines 12, 13).

We consider the effect of $T_{\mathcal{P}}$-Unfolding on the program from our running example.

Example 57 (cont'd)**.** *First, recall that choice constraints* $\{a\} \leftarrow$ *are a shorthand for* $a \leftarrow notna$ *and* $na \leftarrow nota$*. We compute* $T_{\mathcal{P}}$-UNFOLD(Π_{sm}, s) *using the unfolding sequence*

$$
\begin{aligned}
s &= s^{guess} s^{smokes} \\
s^{guess} &= st(1)nst(1) \dots st(3)nst(3)inf(3,1)ninf(3,1) \dots inf(2,3)ninf(2,3) \\
s^{smokes} &= sm(1)sm(2)sm(3)sm(1)sm(2)
\end{aligned}
$$

and obtain:

$$
\begin{array}{rlrl}
\{st(1)\} & \leftarrow & \{st(2)\} & \leftarrow \quad \{st(3)\} \leftarrow \\
\{inf(3,1)\} & \leftarrow & \{inf(1,2)\} & \leftarrow \quad \{inf(2,3)\} \leftarrow \\
sm(1)^1 & \leftarrow st(1) & sm(1)^1 & \leftarrow inf(3,1), \bot \\
sm(2)^1 & \leftarrow st(2) & sm(2)^1 & \leftarrow inf(1,2), sm(1)^1 \\
sm(3) & \leftarrow st(3) & sm(3) & \leftarrow inf(2,3), sm(2)^1 \\
sm(1) & \leftarrow st(1) & sm(1) & \leftarrow inf(3,1), sm(3) \\
sm(2) & \leftarrow st(2) & sm(2) & \leftarrow inf(1,2), sm(1)
\end{array}
$$

We observe that $T_{\mathcal{P}}$-Unfold$(.,s)$ *is faithful for* Π_{sm}.

The output of $T_{\mathcal{P}}$-Unfold(Π, s) is always an acyclic program, however, the faithfulness for Π depends on s. To ensure that $T_{\mathcal{P}}$-Unfold$(.,s)$ is faithful, it is enough to iterate over all variables $n = |\mathcal{A}(\Pi)| + 1$ times, since every derivation in Π can only take n steps. However, as we have seen in the previous example, it can be sufficient to use much fewer steps. This is because the number of times a variable needs to be considered in an unfolding sequence depends on the positive dependencies involving it: e.g. $\mathrm{st}(i), i = 1, \ldots, 3$ does not positively depend on any variable and can thus be considered once before all other variables and never again afterwards.

We give a sufficient condition for faithfulness. Notably, our condition abstracts away the actual program Π and is based on a structural property of the *digraph unfolding* of DEP(Π).

Definition 156 (Digraph Unfolding)**.** *Let G be a digraph and let $s \in V(G)^*$ be an unfolding sequence. Let $cnt(a,s) = |\{j \mid s_j = a\}|$ be the number of occurrences of a in the unfolding sequence s, then the unfolding $UF(G,s)$ of G with respect to s is the digraph U such that*

(i) $V(U) = \{a^i \mid 1 \leq i \leq cnt(a,s)\}$, *and*

(ii) $(b^i, a^j) \in E(U)$ *if* $(b,a) \in E(G)$, $cnt(a, s_1 \ldots s_k) = j$ *for some* k *and* $cnt(b, s_1 \ldots s_k) = i > 0$.

The idea is that the digraph unfolding of DEP(Π) with respect to $s \in \mathcal{A}(\Pi)^*$ is the dependency graph of the unfolded program $T_{\mathcal{P}}$-Unfold(Π, s). Therefore, the vertices are the copies of the atoms (see lines 6,8) and there is an edge (b^i, a^j) if b^i is the last copy of an atom that is used to derive a^j, i.e. the j-th occurrence of a in s (see lines 10,11). Formally:

Lemma 157. *Let Π be an answer set program and $s \in \mathcal{A}(\Pi)^*$ be an unfolding sequence. Then $UF(DEP(\Pi), s) = DEP(T_{\mathcal{P}}\text{-Unfold}(\Pi, s))$ (when identifying a with $a^{cnt(a,s)}$).*

For the proof see Appendix C.2.

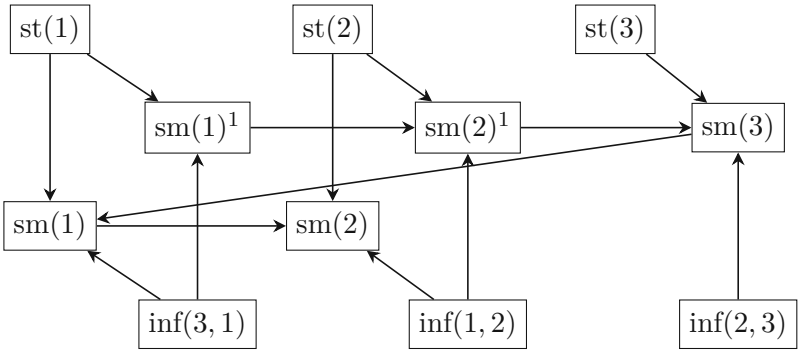


Figure 4.7: Dependency Graph of $\mathrm{T}_{\mathcal{P}}$-Unfold(Π_{sm}, s).

Example 58 (cont'd)**.** *The dependency graph of $\mathrm{T}_{\mathcal{P}}$-Unfold(Π_{sm}, s) is given in Figure 4.7. It is acyclic and corresponds to $UF(DEP(\Pi_{sm}), s)$ as expected.*

Using Lemma 157, we can provide a sufficient condition for faithfulness.

Theorem 158. *Let Π be an answer set program and $s \in \mathcal{A}(\Pi)^*$ be an unfolding sequence. If for every simple directed path $\pi = (a_1, \ldots, a_n)$ in $DEP(\Pi)$ some directed path $\pi_c = (a_1^{c_1}, \ldots, a_n^{c_n})$ in $UF(DEP(\Pi), s)$ exists, then $\mathrm{T}_{\mathcal{P}}$-Unfold$(., s)$ is faithful for Π.*

Proof. Let Π be an answer set program, s an unfolding sequence that satisfies the precondition of the theorem, and $\mathcal{I} \subseteq \mathcal{A}(\Pi)$. We know that for all s the reduct $\mathrm{T}_{\mathcal{P}}\text{-Unfold}(\Pi, s)^{\mathcal{I}_{ext}}$ for $\mathcal{I}_{ext} \cap \mathcal{A}(\Pi) = \mathcal{I}$ is $\mathrm{T}_{\mathcal{P}}\text{-Unfold}(\Pi, s)^{\mathcal{I}}$ as the rules added in line 11 use the original negative body $B^-(r)$, which only uses atoms from $\mathcal{A}(\Pi)$. Hence we can consider $\mathrm{T}_{\mathcal{P}}\text{-Unfold}(\Pi, s)^{\mathcal{I}}$, which has a unique minimal model. We see that if $\mathrm{T}_{\mathcal{P}}\text{-Unfold}(\Pi, s)$ has an answer set $\mathcal{I}_{ext}$ equal to $\mathcal{I}$ on $\mathcal{A}(\Pi)$, then it is the only such answer set.

By the same argument we see, that taking the reduct w.r.t. $\mathcal{I}$ and $\mathrm{T}_{\mathcal{P}}$-Unfolding commute: $\mathrm{T}_{\mathcal{P}}\text{-Unfold}(\Pi, s)^{\mathcal{I}} = \mathrm{T}_{\mathcal{P}}\text{-Unfold}(\Pi^{\mathcal{I}}, s)$. Since $\mathcal{I}$ is an answer set iff it is a minimal model of the reduct $\Pi^{\mathcal{I}}$, it remains to show that $a \in \mathcal{A}(\Pi)$ is derivable from $\Pi^{\mathcal{I}}$ iff it is derivable from $\mathrm{T}_{\mathcal{P}}\text{-Unfold}(\Pi^{\mathcal{I}}, s)$. Since both programs are positive, a is derivable iff it has an SLD tree. W.l.o.g. we can assume an SLD tree such that every path from the root to a leaf iS simple. However, we know that s preserves all simple paths and since the paths in every SLD tree correspond to paths in DEP(Π), we know there exists a corresponding SLD-tree in $\mathrm{T}_{\mathcal{P}}\text{-Unfold}(\Pi^{\mathcal{I}}, s)$. □

Note that we can prove a similar result for $\mathrm{T}_{\mathcal{P}}$-compilation, which reaches a fixed point iff $\mathrm{T}_{\mathcal{P}}\text{-Unfold}(., s)$ is faithful for Π.

This theorem gives us a sufficient condition for an unfolding sequence s to lead to faithfulness of $\mathrm{T}_{\mathcal{P}}\text{-Unfold}(., s)$ with respect to a program Π. Notably, the condition

does not depend on the actual program itself but only on its dependence graph and its unfolding with respect to s.

We are not only interested in faithfulness but we also care about the treewidth increase caused by unfolding. We can bound this increase as follows:

Lemma 159. *Let Π be an answer set program with treewidth k and $s \in \mathcal{A}(\Pi)^*$ be an unfolding sequence. If every variable $a \in \mathcal{A}(\Pi)$ occurs at most m times in s, then the treewidth of $T_{\mathcal{P}}$-Unfold(Π, s) is less or equal to $k \cdot m$.*

Proof (Sketch, for the full proof see Appendix C.2). We know that during unfolding we introduce at most $m-1$ copies $a_1, \ldots, a_{m-1}$ of a variable $a \in \mathcal{A}(\Pi)$. Now, let (T, χ) be a tree decomposition for PRIM(Π) of width k. Then (T, χ'), where

$$\chi'(t) = \chi(t) \cup \{a_j \mid 1 \leq j \leq m-1, a \in \chi(t)\}$$

is a tree decomposition of T$_{\mathcal{P}}$-Unfold(Π, s), and $|\chi'(t)| \leq |\chi(t)| \cdot m \leq k \cdot m$. □

We remark that the converse of this Lemma does not hold.

There is another relevant observation that we need to keep in mind when it comes to knowledge compilation later on. While the guarantee generally only gives us an exponential upper bound in terms of treewidth, we actually know that this upper bound is too pessimistic. Consider the following scenario: if we introduce m copies $a, a_1, \ldots, a_{m-1}$ of a variable a using T$_{\mathcal{P}}$-Unfolding, then we have a monotonic relation between the truth values of the copies. That is, if a_i is true in a stable model $\mathcal{I}$ of the program and $j > i$, then also a_j must be true in $\mathcal{I}$. Thus, restricted to the variables $a, a_1, \ldots, a_{m-1}$, there are actually at most m different assignments that can be extended to a stable model. It follows that for the variables in a bag $\chi'(t)$ of the tree decomposition that we constructed in the proof of Lemma 159 only $m^k = 2^{k \cdot \log_2(m)}$ different assignments can be extended to a stable model. We thus believe that knowledge compilation has better performance than Theorem 137 guarantees.

Motivated by Lemma 159 and Theorem 158, we say that an unfolding sequence s is a *path-preserving m-unfolding sequence* (for digraph G), if for every simple (directed) path $\pi = (a_1, \ldots, a_n)$ in G there is a (directed) path $\pi_c = (a_1^{c_1}, \ldots, a_n^{c_n})$ in UF(G, s) and every variable $a \in V(G)$ occurs at most m times in s. Naturally, we are interested in path-preserving m-unfolding sequences for small m.

Since also Lemma 159 does not access the actual program itself but only its dependency graph and the unfolding sequence, we can abstract away the program and search for path-preserving m unfolding sequences on digraphs directly. As we do not want to do a blind search, we instead consider backdoors [FS15] and generalize them to *component-boosted backdoors* into acyclicity because we can give guarantees for the maximum necessary m based on them. Backdoors have already been considered in the context of ASP [FS15]. Usually, the notion of a backdoor for a digraph G is a vertex set S, such that $G \setminus S$ satisfies

some desirable property. For us this property is (directed or undirected) acyclicity of $G \setminus S$, since we can find a suitable unfolding sequence when given S in either case.

Both the backdoor size of a digraph and its generalization to component-boosted backdoor size intuitively measure the cyclicity of the given digraph by asking how many vertices need to be "cut out" in order for the remaining digraph to be (almost) acyclic. When a parameter value is low, an unfolding sequence s exists such that $\mathrm{T}_{\mathcal{P}}$-Unfold$(., s)$ is faithful and every variable occurs only a few times in s.

Definition 160 (Backdoor, Component-boosted Backdoor)**.** *Let G be a digraph. Then $bs(G)$, the* backdoor size *of G is*

$$\min\{|S| \mid S \subseteq V(G), G \setminus S \text{ is a polyforest or acyclic}\},$$

where a polyforest is graph that has no undirected cycles.

Furthermore, $cbs(G)$, the component-boosted backdoor size *of G, is*

(i) *1, if G is acyclic (which includes $V(G) = \emptyset$),*

(ii) *2, if G is a polytree, i.e., a connected polyforest,*

(iii) $\max\{cbs(C) \mid C \in SCC(G)\}$*, if G is cyclic but not strongly connected,*

(iv) $\min\{cbs(G \setminus S) \cdot (|S| + 1) \mid S \subseteq V(G), S \neq \emptyset\}$ *otherwise.*

As the name suggests, *cbs* adds *component-boosting* to a specific variant of *backdoors*. A related parameter is elimination distance [GHN04]. However, for elimination distance the removal set S in (iv) may only contain 1 element and results in cost $\mathrm{cbs}(G \setminus S) + 1$. Thus, the elimination distance to acyclic digraphs and polytrees is bounded iff cbs(.) is bounded, but naïvely, we can only assert that if the elimination distance is k then cbs(.) is less or equal to 2^k.

The main difference compared to the backdoor size is that component-boosted backdoor size additionally takes into account that $G \setminus S$ may consist of separate SCCs that can be handled recursively. We can therefore bound cbs(.) in terms of bs(.) as follows:

Lemma 161. *Let G be a digraph. Then $cbs(G) \leq 2 \cdot (bs(G) + 1)$.*

Proof. If G is acyclic, then

$$\mathrm{cbs}(G) = 1 \leq 2 = 2 \cdot (0 + 1).$$

If G is a polytree, then

$$\mathrm{cbs}(G) = 2 \leq 2 = 2 \cdot (0 + 1).$$

If G is cyclic and strongly connected, let S^* be a set of vertices such that $|S^*| = \mathrm{bs}(G)$ and $G \setminus S^*$ is a polyforest or acyclic. Then

$$\begin{aligned}\mathrm{cbs}(G) &= \min\{\mathrm{cbs}(G \setminus S) \cdot (|S| + 1) \mid S \subseteq V(G), S \neq \emptyset\} \\ &\leq \mathrm{cbs}(G \setminus S^*) \cdot (|S^*| + 1) \\ &\leq 2 \cdot (\mathrm{bs}(G) + 1).\end{aligned}$$

If G is cyclic but not strongly connected, then

$$\mathrm{cbs}(G) = \max\{\mathrm{cbs}(C) \mid C \in \mathrm{SCC}(G)\}.$$

Let S^* be a set of vertices such that $|S^*| = \mathrm{bs}(G)$ and $G \setminus S^*$ is a polyforest or acyclic. For each $C \in \mathrm{SCC}(G)$ it holds that $C \setminus S^*$ is a polyforest of acyclic, thus $\mathrm{bs}(C) \leq \mathrm{bs}(G)$. From the fact that each C is either acyclic, a polytree, or cyclic and strongly connected, it follows by the previous three cases that $\mathrm{cbs}(C) \leq 2 \cdot (\mathrm{bs}(C) + 1) \leq 2 \cdot (\mathrm{bs}(G) + 1)$. □

Example 59 (cont'd)**.** *Consider the dependency graph $DEP(\Pi_{sm})$ in Figure 4.1a. It is strongly connected and not a polytree, therefore $cbs(DEP(\Pi_{sm}))$ is given by case (iv)*

$$\min\{cbs(DEP(\Pi_{sm}) \setminus S)(|S| + 1) \mid S \subseteq V(DEP(\Pi_{sm})), S \neq \emptyset\}.$$

We see that if we take away any $S_i = \{sm(i)\}, i = 1, \ldots, 3$, then $DEP(\Pi_{sm}) \setminus S_i$ is acyclic. It follows that $cbs(DEP(\Pi_{sm})) \leq cbs(DEP(\Pi_{sm}) \setminus S_i) \cdot (|S_i| + 1) = 2$. Since we need to remove at least one element, this is also a lower bound and hence $cbs(DEP(\Pi_{sm})) = 2$.

In this example the upper bound given by the backdoor size and component-boosted backdoor size align. However, for larger, more complex graphs $cbs(.)$ can be much smaller than the upper bound guaranteed by backdoor size.

With the definition of the component-boosted backdoor size in mind, we state the main result of this section.

Theorem 155. *For any factorized measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$, we can construct in polynomial time in the size of Π given access to an NP-oracle a factorized measure $\mu' = \langle \Pi', \alpha, \mathcal{R} \rangle$ with an acyclic program Π' such that*

(i) *for all $a \in \mathcal{A}(\Pi)$ it holds that $\mu(a) = \mu'(a)$,*

(ii) *the treewidth of Π' is at most $k \cdot cbs(DEP(\Pi))$, where k is the treewidth of Π, and*

(iii) *the size of Π' is at most $cbs(DEP(\Pi)) \cdot |\Pi|$.*

Of course it is somewhat undesirable that we need access to an NP-oracle for the construction. However, this is unavoidable, since computing cbs(.) is NP-hard.

Theorem 162. *The problem of checking whether $cbs(G) \leq k$ given a digraph G and $k \in \mathbb{N}$ in the input is NP-complete.*

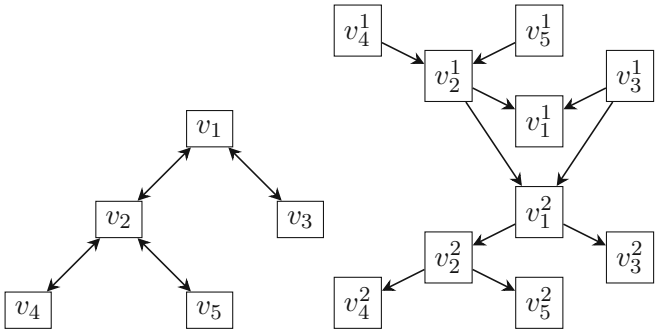


Figure 4.8: A polytree G with root v_1 (left) and the unfolding $\mathrm{UF}(G, s_{post}s_{pre})$, where $s_{post} = v_4v_5v_2v_3v_1, s_{pre} = v_1v_3v_2v_5v_4$ (right).

Proof (Sketch, for the full proof see Appendix C.2). NP-membership is easy to see by a guess and check algorithm.

For NP-hardness, we use a reduction from SAT by constructing a digraph G such that its backdoor size is k iff the SAT instance is solvable. Then we modify the digraph in such a manner that backdoor size and component-boosted backdoor size are equal to obtain the desired result. □

We will see later on that luckily the NP-hardness does not cause any problems in practice.

To prove Theorem 155, we show that every digraph G has some path-preserving $\mathrm{cbs}(G)$-unfolding sequence, using structural induction on the definition of cbs(.).

Lemma 163. *Let G be an acyclic digraph. Then there exists a path-preserving 1-unfolding sequence. For case (i), we obtain:*

Proof. Let s be an unfolding sequence where every $a \in V(G)$ occurs exactly once and which obeys a topological ordering of G. Then $\mathrm{UF}(G, s)$ is equal to G (modulo variable renaming) and therefore path-preserving. □

Next, we consider case (ii), where G is a polytree.

Lemma 164. *For every polytree G there exists a path-preserving 2-unfolding sequence s.*

Proof. As G is a polytree, the corresponding undirected graph G^{tree} is a tree with some arbitrarily chosen root. Let $s_{post}, s_{pre} \in V(G)^*$ be sequences such that every vertex occurs in s_{post} and s_{pre} after all its descendants and ancestors in G^{tree}, respectively. Then the concatenation $s_{post}s_{pre}$ of s_{post} and s_{pre}, is a path-preserving 2-unfolding sequence of G, as depicted in Figure 4.8. □

In case (iii), which is the first recursive one, we assume that G is cyclic but not strongly connected. Here, we divide the problem into one subproblem for each SCC of G and obtain a global solution by combining the solutions for the subproblems.

Lemma 165. *Let G be a cyclic but not strongly connected digraph, and for each $C \in SCC(G)$ let $s_C \in V(C)^*$ be a path-preserving $cbs(C)$-unfolding sequence for C. Then some path-preserving $cbs(G) = \max_{C \in SCC(G)} cbs(C)$-unfolding sequence for G exists.*

Proof. Let G^{con} be the condensation of G, i.e. $V(G^{con}) = \mathrm{SCC}(G)$ and $(C, C') \in E(G^{con})$ if there exist $v \in V(C), v' \in V(C')$ such that $(v, v') \in E(G)$. Since G^{con} is acyclic we can assume a topological order $(C_1, \ldots, C_n)$ of G^{con} to be given. Consider, $s = s_{C_1} \ldots s_{C_n}$, the concatenation of the unfolding sequences for the SCCs in the chosen topological order. It is a path-preserving unfolding sequence for G since for every directed simple path in G that contains $a, b \in V(G)$ it holds that if $a \in C_i$ and $b \in C_j$ such that $i < j$ then a must occur after b. Therefore, as the sequences s_{C_i} per component are path-preserving, we know that the whole sequence is path-preserving. Furthermore, since $V(C_i) \cap V(C_j) = \emptyset$ for $i \neq j$ and $s_{C_i} \in V(C_i)^*$, it is clear that the maximum number of times a vertex $a \in V(G)$ occurs in s is bounded by $\max_{C \in \mathrm{SCC}(G)} \mathrm{cbs}(C) = \mathrm{cbs}(G)$. □

Last but not least, we consider case (iv), the second recursive case. Here, G is strongly connected but not a polytree. We remove a set $S \subseteq V(G)$ of "problematic" vertices such that the component-boosted backdoor size of the rest, i.e., $\mathrm{cbs}(G \setminus S)$, is small and handle S and $G \setminus S$ separately.

Lemma 166. *Let G be a strongly connected digraph, $S \subseteq V(G)$ and $s_r \in V(G \setminus S)^*$ a path-preserving m_r-unfolding sequence. Then there exists a path-preserving $m_r(|S|+1)$-unfolding sequence for G.*

Proof. Let $S = \{a_1, \ldots, a_{|S|}\}$. We define $s_S = a_1 \ldots a_{|S|}$ and $s = (s_r s_S)^{|S|} s_r$, i.e., $s \in V(G)^*$ is the sequence obtained by iterating $|S|$ times over the sequence $s_r s_S$ and finally concatenating s_r. Then s is an $m_r(|S|+1)$-unfolding sequence, as every $a \in S$ occurs exactly $|S| \leq m_r(|S|+1)$ times, and every $a \in V(G \setminus S)$ occurs at most m_r times in s_r and at most $m_r(|S|+1)$ times in general.

Furthermore, s is path-preserving: every simple directed path π in G uses $k \leq |S|$ vertices from S and thus $\pi = \pi_1, a_{i_1}, \pi_2, a_{i_2}, \ldots, a_{i_k}, \pi_k$, where π_i is a simple directed path in $G \setminus S$. Consider Figure 4.9, which sketches $\mathrm{UF}(G, s)$ and the path π. As s_r is path-preserving for $G \setminus S$, we know that we can walk π_1 in $\mathrm{UF}(G \setminus S, s_r)$, then go to $a_1 \in S$, walk the path π_2 in $\mathrm{UF}(G \setminus S, s_r)$ and so on. □

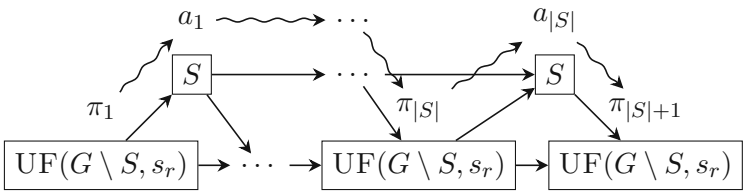


Figure 4.9: Sketch of $\mathrm{UF}(G, s)$ and a path π through it, for the second recursive case, as in the proof of Lemma 166.

4.7 Implementation

In the previous sections, we discussed the general pipeline for solving AASC problems, and we considered for different aspects such as the knowledge compilation, the cycle breaking and the Clark Completion which options are available and which theoretical guarantees we can give for them. In the following, these theoretical considerations will serve as a basis for the choices we make when implementing our approach to solving AASC problems in our open-source solver aspmc[5].

As other available solvers, aspmc follows the general pipeline given in Figure 4.3. In contrast to some of them, aspmc does not skip any of its steps. In the following, we discuss the implementation aspects of each of the steps both technically and in relation to the theoretical results discussed above.

4.7.1 Input Specification

For the inputs we do not allow general algebraic measures but restrict ourselves to the fragment of programs that can be specified similarly to ProbLog syntax. From an implementation perspective, it would also be easily possible to accept algebraic measures in their general form, however we did not deem this necessary, since Theorem 131 shows that this is not a restriction in terms of expressivity. We chose this syntax, since it already well-known from ProbLog and allows ProbLog, aspmc, and PITA to be used interchangeably.

More formally, we allow programs consisting of rules that are either standard normal ASP rules, or rules of the form

$$r_1 :: a_1; \ldots; r_k :: a_k \text{ :- } b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m. \tag{4.21}$$

Here, $r_1, \ldots, r_k$ are semiring values over the specified semiring and all a_i, b_j, and c_l are atoms. By default, the probabilistic semiring $\mathcal{P}$ is assumed, other semirings can be given as an input argument, specifying either one of the standard semirings that is included

[5] aspmc can be installed as a python package from `https://pypi.org/project/aspmc/` for easy use and its source code can be downloaded from `https://github.com/raki123/aspmc/` for development purposes.

in aspmc or a non-standard semiring that is specified as a python module, which is dynamically loaded based in its name.

A rule of the form (4.21) specifies that if the body is satisfied, then we can derive up to one of the atoms $a_1, \ldots, a_k$. Then, if we use the rule to derive a_i then the rule contributes a factor of r_i to the weight of the model. If the body of the rule is satisfied but none of the atoms $a_1, \ldots, a_k$ are derived by it, then the rule contributes a factor of $\text{negate}(r_1 \oplus \ldots \oplus r_k)$ to the weight of the model. Here, negate is a function defined by the semiring module. E.g., for the probabilistic semiring it makes sense to define $\text{negate}(p)$ as $1 - p$ in order to ensure that the overall probability mass defined by a program is 1. Another generally applicable option is to specify $\text{negate}(r)$ as $e_\oplus$ to ensure that exactly one of the a_i is derived if the body of the rule is satisfied, since otherwise the weight of the corresponding model is zero.

Furthermore, atom queries are specified immediately within the program via statements of the form

$$query(a).$$

for an atom a. A program can contain multiple such statements, which then are each evaluated as the weight of all answer sets such that a holds.

We want to stress that the rule only contributes a factor r_i, if it was used to derive a_i, rather than if its body is satisfied and a_i is true. This has important implications, such as the following:

Example 60 (Two Possible Derivations)**.** *Consider the following program over the probability semiring $\mathcal{P}$:*

$$0.5 :: a. \qquad 0.5 :: a. \qquad query(a).$$

This program specifies that there are two independent *ways to derive a and we are interested in the weight of the answer sets such that a holds. We can derive a either only via the first rule, only via the second rule, via both rules or not at all. Each of these possibilities leads to an answer set with probability* 0.25*. Thus, the result of the query for a is* 0.75*. Clearly, the program*

$$0.5 :: a. \qquad query(a).$$

would instead only lead to a result of 0.5*.*

This behavior may seem unintuitive at first glance, however, it makes much more sense when we interpret rules of the form 4.21 as a possible reason that can cause a_i. If the rule always contributed a weight r_i, when a_i and its body hold, then the program

$$a. \qquad 0.5 :: a. \qquad query(a).$$

would lead to a query result of 0.5 although clearly a has a 100% probability of being true in a randomly selected answer set.

We allow non-ground programs, which may even have variables in the place of semiring values but do not go into the details here for simplicity.

In order to parse programs that are specified using the above syntax, we wrote our own parser in python using the parser generator lark [17], which is able to generate an LL parser on the fly given its specification as a grammar. The generation on the fly is important, as we need to incorporate parsing of semiring values dynamically based on the semiring we use.

4.7.2 Grounding & Simplification

After parsing the possibly non-ground input specification, we need to ground the program in a first step. There exist a variety of grounding engines for programs already, for example, in ProbLog or clingo. Neither of them accept exactly the same kind of programs as we do, nevertheless, we were still able to exploit the grounding capabilities of clingo, instead of reinventing the wheel. For this, we restructure the parsed input as follows.

Standard ASP rules are kept as they are. A rule of the form

$$r = r_1 :: a_1; \ldots; r_k :: a_k \text{ :- } b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m.$$

is transformed into the set of rules

$$\begin{aligned} guess(r) \text{ :- } & b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m. \\ a_1 \text{ :- } & b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m, \text{alg}(id, 1, \text{var}(r), a_1, r_1). \\ & \ldots \\ a_k \text{ :- } & b_1, \ldots, b_n, \text{not } c_1, \ldots, \text{not } c_m, \text{alg}(id, k, \text{var}(r), a_k, r_k). \end{aligned}$$

where $guess(r)$ is

$$\text{alg}(id, 1, \text{var}(r), a_1, r_1); \ldots; \text{alg}(id, k, \text{var}(r), a_k, r_k); \text{alg}(id, k+1, \text{var}(r), n, n).$$

Here, an atom of the form $\text{alg}(id, i, \text{var}(r), a_i, r_i)$ for $i = 1, \ldots, k$ is true, when the rule with id id is used to derive the atom a_i with weight r_i, instantiated with the variables in $\text{var}(r)$. Thus, the first rule states for each of the instantiations of the variables $\text{var}(r)$ of r that if the body holds, then we may derive one of the atoms a_i. If $\text{alg}(id, k+1, \text{var}(r), n, n)$ is true, then none of the atoms are supposed to be derived.

The remaining rules then implement that if the body holds and the atom $\text{alg}(id, i, \text{var}(r), a_i, r_i)$ is set to true, then a_i is derived.

If the semiring is the probabilistic semiring, we in addition apply some further optimizations of the ProbLog solver. Since they are described in detail in Appendix B of [DK15], we do not discuss them here and only note that they seem to be beneficial for performance but do not trivially generalize to other semirings.

Notably, clingo not only grounds programs but performs basic simplifications, such as removing atoms that are stated as facts from the bodies of rules. Apart from this, the

only additional simplification that we perform after grounding is that we omit rules whose head atom occurs positively in the body. There are other simplifications that one could apply, such as the restriction of the program to its relevant ground part or the replacement of algebraic atoms a with weight $e_\oplus$ by the constraints `: - a.`. The former has so far only been considered for probabilistic reasoning and is implemented in ProbLog. We did not implement the restriction to the relevant ground part yet, however if this capability is desired, ProbLog can be used as a preprocessor. We leave generalized and possibly extended preprocessing for future work.

4.7.3 Cycle Breaking

Recall that the three approaches to eliminate cycles that we considered in Section 4.6 have the following properties:

- MJ(.) is implemented in ProbLog, has a treewidth upper bound of $\mathcal{O}(k \cdot s \cdot c)$, where k is the original treewidth s is the size of the largest SCC of the dependency graph and c is the maximum number of simple cycles in an SCC of the dependency graph.
- JN(.), which has been considered for ProbLog [Fie+11] (an implementation is available online [06]) comes instead with a treewidth upper bound of $\mathcal{O}(k \log(s))$.
- Our $\mathrm{T}_\mathcal{P}$-Unfolding algorithm has a treewidth upper bound of $\mathcal{O}(k \cdot \mathrm{cbs}(\mathrm{DEP}(\Pi)))$, where cbs(.) is a structural parameter that intuitively measures the cyclicity of the dependency graph $\mathrm{DEP}(\Pi)$ of the program Π.

Judging by the treewidth guarantee alone, it suggests itself to use JN(.), since its treewidth increase scales logarithmically with the size of the largest SCC. Thus, unless the program dependency graph $\mathrm{DEP}(\Pi)$ is rather sparse, we expect that $\mathrm{cbs}(\mathrm{DEP}(\Pi))$ supersedes $\log_2(s)$. However, as we mentioned above, JN(.) uses binary counters in the encoding, which may impact negatively the performance during solving.

Furthermore, Fierens et al.'s [Fie+11] comparison of MJ(.) and JN(.) provides additional interesting aspects to take into consideration. They found that on small programs MJ(.) produces significantly smaller CNFs than JN(.), while JN(.) leads to much smaller CNFs on larger programs. This suggests that MJ(.) allows for faster inference on small programs. Furthermore, they found that only very few programs seem to be large enough to make JN(.) profitable, meaning that knowledge compilation is feasible in principle and JN(.) yields better results. While the state of the art in knowledge compilation has seen significant improvements since 2011, we expect that both binary counters and significant size differences still will play a role in addition to treewidth.

This brings us to the $\mathrm{T}_\mathcal{P}$-Unfolding approach. It is an improvement upon MJ(.) both in terms of the size of the encoding, which is at most quadratic with a small constant factor, and in terms of the treewidth, which is at most linear in the size of the largest SCC S even

if S is fully connected. We thus expect that $T_{\mathcal{P}}$-Unfolding significantly outperforms both MJ(.) and JN(.), even when using the current state of the art Knowledge Compilers.

In order to perform $T_{\mathcal{P}}$-Unfolding by Algorithm 2, we need a path preserving unfolding sequence s though. We have shown that based on a small component-boosted backdoor, which is guaranteed to exist if cbs(.) is small, we can compute a good such sequence s. However, computing this parameter exactly is NP-hard (see Theorem 162). While the backdoor size is also NP-hard to compute, there exist efficient solver for it. Thus, we restrict ourselves to computing a (small) backdoor into polytrees for every SCC S of DEP(Π), i.e., a subset $B \subset S$ such that no undirected cycle in DEP(Π) lies in $S \setminus B$.

Note that a backdoor into polytrees is a Feedback Vertex Set (FVS). Computing a FVS is a well studied problem and there are many open source solvers available due to the 2016 edition of PACE [Del+16]. We use the implementation [KP18b] of Kiljan and Pilipczuk [KP18a], who re-implemented many of the participating algorithms under a permissive license. We run the exact version of the solver using a fixed timeout of 30 seconds to try to obtain an optimal backdoor B. If the answer is not produced within the timeout, we run the heuristic version of the solver to obtain an approximation quickly.

4.7.4 Clark's Completion

After performing $T_{\mathcal{P}}$-Unfolding, the resulting program is tight, i.e., its dependency graph is acyclic. Therefore, we can apply Clark's Completion on it to receive a CNF that we can then compile in the next step. Here, we again have three possible options:

- Clark(.), which does not admit any treewidth guarantees;
- PClark(.), i.e., Hecher's [Hec22] primal tree decomposition guidance, which admits a treewidth guarantee of $3(k+1)$ if k is the treewidth of the tree decomposition of PRIM(Π) in use;
- IClark(.), i.e., our novel incidence tree decomposition guidance, which admits a treewidth guarantee of $3(k+1)$ if k is the treewidth of the used tree decomposition of INC(Π) in use.

Based on the treewidth guarantees alone, it is suggestive to always use IClark(.). However, while it has the best worst case guarantees, it is not guaranteed that the obtained CNF always has the lowest treewidth out of the three. While the resulting treewidth cannot become much higher than that of a CNF obtained by Clark(.) or PClark(.), it may be as high or slightly higher and lead to a slightly larger CNF.

We therefore implemented all three versions of Clark's Completion and allow for a selection via input parameters, which we will use to compare the different strategies below.

In order to apply the tree decomposition guided versions PClark(.) and IClark(.) of Clark's Completion, we need access to tree decompositions of PRIM(Π) and INC(Π).

Generating optimal tree decompositions is expensive but there exist several good heuristic solvers such as flow-cutter [HS18], tamaki-2017 [Tam19], and htd [AMW17]. We use flow-cutter since it performed very well in the PACE competition of 2017 regarding the heuristic computation of treewidth, achieving the best average approximation ratio and the best maximum approximation ratio of all solvers [Str17]. Furthermore, it has a permissive open source license that allows for combining it with closed source code such as c2d (version 2.20). Whenever we generate tree decompositions, we use a given timeout from the input. We wait until a tree decomposition has been found and the timeout has passed, whichever happens later. Then we use the best tree decomposition found. This means that on large graphs we may supersede the given timeout.

After Clark's Completion, we extend the resulting CNF with annotations denoting the semiring in use, and the weights of literals over the semiring.

4.7.5 Knowledge Compilation

For the knowledge compilation step, there is a wide range of tools available that compile to different tractable circuit representations, such as

- to BDDs, e.g., [Lin99; Som12]
- to SDDs, e.g., [Dar11], and
- to d-DNNFs, e.g., [Dar04; LM17; Mui+12; OD15].

We focus on the compilers for d-DNNFs, as they tend to have the best worst-case guarantees in terms of treewidth. Furthermore, they also usually perform well on the related tasks of model counting and weighted model counting as seen in the Model Counting Competition 2020 [FHH21].

More specifically, we implemented the knowledge compilation step for solvers that are recent or ranked high in the 2021 edition of the weighted model counting track of the Model Counting Competition 2021[Joh21]. We focus on this track, as we are interested in the performance of the actual compilation. The unweighted track could possibly have skewed results as preprocessing that is not applicable to weighted can lead to significant performance improvements. The solvers sharpSAT-TD [KJ21], d4 [LM17], and c2d [Dar04] finished top three in the weighted model counting track, miniC2D [OD15] is a recent knowledge compiler, which is why we also included it.

Notably, whereas c2d, miniC2D, and d4 allow for knowledge compilation by default, this is not the case for sharpSAT-TD, which was originally a solver conceived only for (weighted) model counting. Thus, we modified sharpSAT-TD to enable its usage for knowledge compilation. This turned out to be rather easy: since sharpSAT-TD was designed to be extensible to general semirings, we could exploit the idea of the arithmetic circuit semiring from Section 4.3.1. Specifically, in our implementation we used d-DNNF nodes as semiring elements and combined them in a conjunctive/disjunctive manner

whenever sharpSAT-TD multiplied/added two values. Not only does this lead to an algebraic circuit but to a d-DNNF, which is moreover smooth[6].

Apart from sharpSAT-TD, we also performed a minor modification in d4. Namely, by default d4 does not produce smooth d-DNNFs, which leads to an increase in the evaluation time. For this reason, we made minor modifications to the part of the source code of d4 that writes d-DNNFs, to ensure their smoothness d-DNNFs. Notably, the necessary information was already present meaning that we could keep our modifications to a minimum[7]. The source code of c2d and miniC2D were left unmodified.

The exact input arguments that we use for the different knowledge compilers can be found in D.1. Most settings are rather standard, however, we should mention that for c2d and miniC2D we supply custom dtrees and vtrees, respectively, which determine the order in which variables are decided during compilation.

We chose to generate custom dtrees and vtrees, since Korhonen and Järvisalo [KJ21] showed that the performance of c2d and miniC2D benefits from it. Namely, they used the fact that, for a given tree decomposition, we can generate dtrees and vtrees that give us a performance guarantee[8] of $\mathcal{O}(2^k \cdot k \cdot |C|^c)$, where k is the width of the tree decomposition, $|C|$ is the size of the CNF, and c is some constant. This guarantee can be achieved by first deciding all variables in the root of the tree decomposition and proceeding recursively for the children [KJ21]. Both c2d and miniC2D were shown to solve significantly more instances using this strategy than when using the standard settings [KJ21]. We provide details regarding how we generate dtrees and vtrees in D.2, which only insignificantly differs from Korhonen and Järvisalo's [KJ21] method. The tree decompositions used in this step are again generated using flow-cutter [HS18] using a timeout given in the input.

4.7.6 Evaluation

Given a (smooth) d-DNNF, the evaluation procedure is rather standard. We initialize the weights of the literals using the weight labels of our extended CNF format. Then, we parse the d-DNNF line by line and interpret it as an arithmetic circuit by combining the inputs to AND gates via multiplication and the inputs to OR gates via addition. The result we obtain at the root node of the d-DNNF then corresponds to the final result of our query. This step is well-known and well-understood, which is why we do not go into details here but refer the interested reader to [KVD17].

[6]Source code of the modified version: `https://github.com/raki123/sharpsat-td`; For the original: `https://github.com/Laakeri/sharpsat-td`.

[7]Source code of the modified version: `https://github.com/raki123/d4`; For the original: `https://github.com/crillab/d4`.

[8]Strictly speaking, it is not clear whether this is only a guarantee for miniC2D but also for c2d at least in the way that we as well as Korhonen and Järvisalo [KJ21] generate dtrees and run c2d. This is because c2d has an optimization built in that skips deciding variables whose "turn" it would usually be, if these variables are not necessary to decompose the current CNF into separate components. This optimization can be turned off using the input option "force" of c2d. However, this seems to only degrade the overall performance.

The only additional comment in order regards the evaluation of non-smooth d-DNNFs as produced by miniC2D, since not all variables are guaranteed to occur in every branch of the d-DNNF, which can lead to wrong results during naive evaluation as described above. The only case in which a literal may not occur in a branch is if both values are accepted in the branch. E.g. for the program $\{a\} \leftarrow$ the empty NNF is a valid d-DNNF. If the variable a does not occur in a branch, we need to multiply the value of the branch by the sum of the weights of a and $\neg a$. Naively, this is easily possible by tracking the assigned and unassigned variables that occur below each node but costly. However, given that we know the vtree that the CNF was compiled with, we can do this more efficiently by keeping track of which vtree node a d-DNNF node belongs to and multiplying its weight by the corresponding factor when we notice that variables are missing.

4.8 Experimental Evaluation

In order to evaluate the impact of our theoretical results in practice, we performed an extensive experimental evaluation. Furthermore, we compared the performance of our solver aspmc to state of the art solvers on probabilistic inference instances. All results and benchmarks can be found online.[9]

4.8.1 Questions & Hypotheses

The first question we consider addresses the properties of the encodings that different cycle breakings result in.

Q1. CNF Encodings How do the cycle breakings MJ(.), JN(.) and T$_\mathcal{P}$-Unfolding(.) differ in terms of *size* and *treewidth (upper bound)* of the final CNF encoding?

We expect that T$_\mathcal{P}$-Unfolding is almost always better than MJ(.) with respect to both aspects (see Section 4.6.4). Furthermore, we expect that for small-sized and medium-sized programs, T$_\mathcal{P}$-Unfolding results in lower CNF sizes than JN(.), but yields larger sizes for bigger programs due to the asymptotic guarantee. In terms of treewidth, we expect that JN(.) leads to lower treewidth unless the given program has only very minor cyclic positive dependencies, which means a very small component-boosted backdoor size.

Second, we are interested in finding the best configuration of options to use for aspmc. While we always use T$_\mathcal{P}$-Unfolding for cycle breaking, we still have to decide which version of Clark's Completion to apply and which knowledge compiler to use.

Q2.1. Variants of Clark's Completion Does tree decomposition-guidance of Clark's Completion provide a benefit compared to unguided Clark's Completion?

[9] github.com/raki123/aspmc_benchmarks/tree/aspmc_results

Q2.2. Effect of Knowledge Compiler Which knowledge compiler leads to the best AASC performance?

For Q2.1, we anticipate that tree decomposition-guidance can only decrease the performance, if the treewidth of the CNF obtained with guidance is the same or slightly higher than the one obtained without guidance. Even then, the decrease should be marginal. However, when the guidance has a positive impact on the treewidth of the resulting CNF, we naturally expect a performance increase.

Regarding Q2.2, it seems reasonable to assume that SHARPSAT-TD performs best, followed by d4, and c2d since they were the top three solvers on the weighted model counting track of the Model Counting Competition 2021. miniC2D did not participate. With Q2.2 we want to find out, whether this translates to the same performance when we perform knowledge compilation first and then evaluate rather than performing weighted model counting immediately within the solver.

Last but not least, given the overall best configuration for AASC with aspmc from the results of the previous questions, we want to compare our solver with the state of the art software in probabilistic logic programming.

Q3. Overall Performance Does aspmc provide a significant performance increase compared to the state of the art solvers for probabilistic logic programming?

We expect this to be the case, since the pipeline that uses knowledge compilation of CNFs comes with the best performance guarantees. Furthermore, we anticipate our cycle breaking and Clark's Completion to exhibit better solving capacity than other approaches that compile CNF to sd-DNNF. On top of that, we use the state of the art knowledge compilers for CNF to sd-DNNF, which have seen significant performance improvements in recent years.

4.8.2 Setup

Benchmark Instances We use probabilistic logic programming instances for all questions, divided into four sets of instances:

- **Smokers** [Fie+15] For programs with varying levels of positive cyclic dependencies, we use the well-known smokers example (see Example 39) with varying extensions of the input predicates person and friend. We generated the benchmarks ourselves using the Barbàsi-Albert graph model [AB01], which is known to generate so called *scale-free* graphs that resemble typical networks in real life. For each vertex in the graph, we add one person and for each edge (v, v') we add friend(v, v') to the input. We used varying graph sizes $(n = 3, \ldots, 49)$ and vertex degrees $(m = 2, \ldots, \min(n-1, 10))$ leading to graphs of varying density (namely, $(n-1) \cdot m$ edges).

- **Near Tree** Additionally, we introduce what we call *near tree* instances: a new cyclic benchmark set concerning reachability on almost trees. To generate the programs, we used as ingredients a directed graph G and the base program

$$\begin{aligned} r(s). \\ 0.1 :: trap(Y) \text{ :- } p(X,Y). \\ r(Y) \text{ :- } p(X,Y). \\ 1/d(X) :: p(X,s_1(X)); \ldots; 1/d(X) :: p(X,s_d(X)) \text{ :- } r(X), \text{not } trap(X). \end{aligned}$$

  Here, $d(X)$ is the number of outgoing arcs of X in G, and the vertices $s_1(X), \ldots, s_d(X)$ are the immediate descendants. We obtain the final program by replacing the variables X, Y with constants corresponding to the vertices of G.

  This program models that we reach (denoted by $r(.)$) the starting vertex s and, at each vertex v that we reach, decide uniformly at random which outgoing arc we include in our path (denoted by $p(.,.)$). If we include the arc (v,w), then we reach the vertex w. However, we only include an arc, if we do not get trapped (denoted by $trap(.)$) at v.

  As for the digraph G, we used two parameters $n, k \in \mathbb{N}$ to generate it as follows. We first generated a random tree of size n using the python library networkx, where each arc in the tree is bidirectional. Then, we added k vertices and connected them to each of the n original vertices in the tree bidirectionally. Finally, we added one vertex as the goal vertex, with incoming arcs from each of the k additional vertices. As the start s we use the root of the tree.

  These graphs are almost trees in the sense that (i) their treewidth is approximately $\min(k,n)$[10] and (ii) its component-boosted backdoor size is $2 \cdot k$, since we obtain a (poly)tree after removing the k added vertices.

  We use one instance for each combination of $n = 10, 20, \ldots, 100$ and $k = 1, 2, \ldots, 5$.

- **Growing Heads & Growing Negated Bodies** [Sht+14] The synthetic benchmark sets of growing heads (gh) and growing negated bodies (gnb) compare the difficulty of reasoning in the presence of long heads such as

$$p_1 :: a_1; \ldots; p_i :: a_i \text{ :- } a_{i+1}.$$

  and long negated bodies

$$0.5 :: a_i \text{ :- not } a_{i+1}, \ldots, \text{not } a_{n-1}, a_n.$$

  Here the length of the heads and negated bodies varies from 1 to 25 and 1 to 200 (after 50 only in steps of 10), respectively.

  Both gh and gnb only feature acyclic programs.

[10] Observe that every vertex in the graph has at least degree $\min(n,k)$, which is known to imply treewidth $\geq \min(n,k)$.

- **Blood** [Sht+14] The blood benchmark set consists of 100 instances, which compute probabilities of blood types given the blood types of ancestors. Also the blood benchmarks are all acyclic; they have vastly varying sizes but treewidth at most 10.

Benchmark Platform We ran all solvers on a cluster of 12 nodes, each equipped with two Intel Xeon E5-2650 CPUs with 2.2 GHz and access to 256 GB shared RAM under Ubuntu 16.04.1 LTS powered on kernel 4.4.0-139 with no hyperthreading operating Python 3.7.6. Per instance, we set a memory limit of 32GB and a time limit of 1800 secs on a single core.

Solvers Compared We compare a variety of solvers from probabilistic logic programming, as well as different configurations of our own solver. Namely, we include

- **LP$^{\text{MLN}}$** [LY17] version 1.1: LP$^{\text{MLN}}$ performs probabilistic reasoning by enumeration of all stable models. plingo [Hah+22] is an alternative solver with the same strategy that however only exhibits slightly better performance [Hah+22].
- **ProbLog** [Fie+15] version 2.1.0.42: ProbLog uses the MJ(.) cycle breaking followed by knowledge compilation. It supports different circuit classes for compilation. We include compilation of CNF to sd-DNNF using either DSHARP [Mui+12] (denoted "+ Dsharp") or c2d (denoted "+ c2d") as knowledge compilers, and of the acyclic programs to SDD using PySDD [Mee18; Dar+17] (denoted "-k sdd).
- **PITA** [RS11] version 4.5.0 using the Prolog engine swipl version 8.5.1: PITA compiles cyclic programs to BDD using the CuDD package version 3.0.0 [Som12].
- **lp2lp2** [JN11] version 1.23: LP2LP2 performs the JN(.) cycle breaking. The input options "-g" and "-l" ensure a bijective correspondence between stable models. Here, we do not perform probabilistic inference, as the weights get lost during cycle breaking. Instead we use SHARPSAT-TD afterwards to compile the resulting CNF into an sd-DNNF to count its models.
- **aspmc** (our solver) version 1.0.7. We always use the input option "-dt 10" to specify that tree decompositions should be computed with a timeout of ten seconds.

  To vary the version of Clark's Completion that we use, we specify "-g none", "-g ors", or "-g both" to use unguided, primal tree decomposition guided or incidence tree decomposition guided Clark's Completion.

  To specify the knowledge compiler we use "-k c2d", "-k miniC2D", "-k d4", or "-k sharpsat-td" to use c2d version 2.20, miniC2D version 1.0.0, d4 from `https://github.com/raki123/d4`, or our modified version of SHARPSAT-TD from `https://github.com/raki123/sharpsat-td`.

Comparisons

Q1. CNF Encodings We produce a CNF using the different cycle breaking algorithms of aspmc, ProbLog, and lp2lp2 and extract the size of the encoding (i.e., number of clauses) as well as decent treewidth upper bounds from the first tree decomposition found by flow-cutter.

Notably, since the CNF encodings all use Clark's Completion, the differences in the size of the encoding and treewidth are due to the cycle breaking implemented by the solver. Therefore, we restrict our comparison here to the smokers and the near tree benchmark sets, which contains the instances where cycle breaking is necessary. Furthermore, we only consider instances whose number of clauses was less than 25000 and whose treewidth upper-bound was less than 200, which includes all solved instances apart from very few outliers.

Recall that the asymptotic size guarantees tell us that large enough instances JN(.) is better than $T_{\mathcal{P}}$-Unfolding, which in turn is better than MJ(.). We assume that the instances, where this kicks in, are so large that knowledge compilation is bound to fail, regardless of the encoding. Consequently, we also take into account the runtime of the instances such that we can establish the properties of the encoding on the instances relevant to us, i.e., those that are in principle solvable using the given cycle breaking.

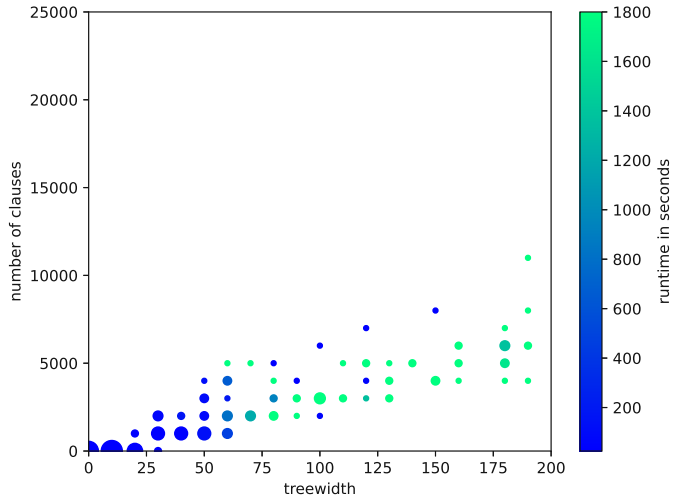
Q2.1. Variants of Clark's Completion Here, we want to ensure that effects on performance for different versions of Clark's Completion results from the exploitation of a lower treewidth in the decision heuristic of the knowledge compiler. Since d4's decision heuristic is not based on tree decompositions and sharpSAT-TD uses a secondary heuristic in combination with the tree decomposition, there is no guarantee that variables are only decided based on the order entailed by the tree decomposition. On the other hand, for c2d and miniC2D, we always specify an input dtree/vtree such that the variables are decided in the order corresponding to the tree decomposition. Therefore, we ran aspmc with varying versions of Clark's Completion and c2d respectively miniC2D as knowledge compiler for this question.

Q2.2. Effect of Knowledge Compiler To test the performance of the different knowledge compilers, we ran aspmc with the best performing version of Clark's Completion from Q2.1.

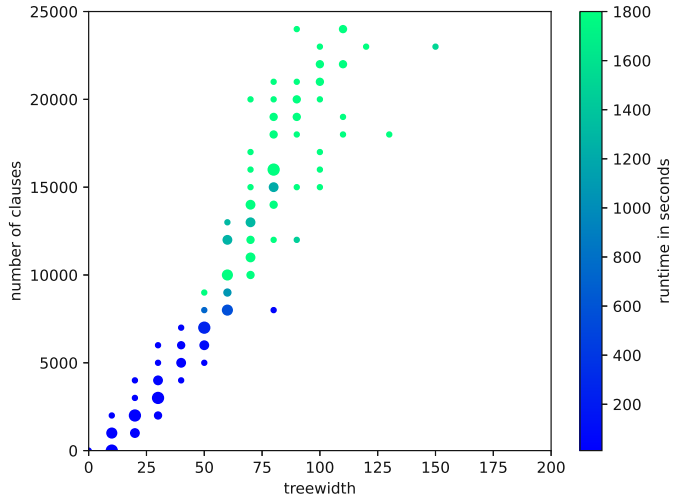
Q3. Overall Performance For the general comparison, we include all solvers and configurations listed above, while for aspmc we use only the best performing Clark Completion.

4.8.3 Results & Discussion

Q1. CNF Encodings The results for this experiment are shown in Figure 4.10, which contains one XY-plot for $T_{\mathcal{P}}$-Unfolding, JN(.), and MJ(.) in Figures 4.10a, 4.10b, and 4.10c, where the runtime is color-coded. We note that in almost all cases a light green



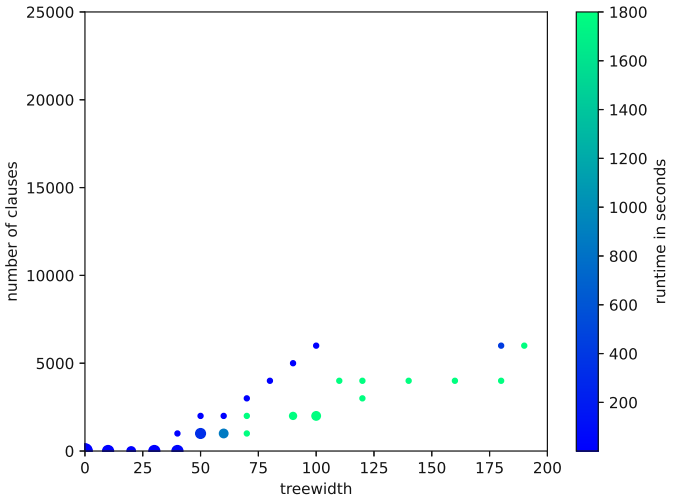
(a) Numbers of clauses, treewidth upper bounds, and average runtime, when using $T_{\mathcal{P}}$-Unfolding (i.e., aspmc) for cycle breaking.



(b) Numbers of clauses, treewidth upper bounds, and average runtime, when using JN(.) (i.e., LP2LP2) for cycle breaking.

data point not only indicates a high running time, close to 1800 seconds, but even that the instances represented by that data point all timed out.

As expected, we see that for MJ(.) very few CNF encodings are in the range of 0 to 25000 clauses and 0 to 200 treewidth upper-bound compared to $T_{\mathcal{P}}$-Unfolding and JN(.), and even fewer are solvable within the time limit shown by the colors of the data points. Both $T_{\mathcal{P}}$-Unfolding and JN(.) have significantly more CNF encodings within the plotted range including also many more solved instances. Interestingly however, the distribution



(c) Numbers of clauses, treewidth upper bounds, and average runtime, when using MJ(.) (i.e., ProbLog) for cycle breaking.

Figure 4.10: The number of clauses, treewidth upper-bounds, and average runtimes of CNF encodings produced by different cycle breaking algorithms on the smokers and tree benchmark set. Restricted to instances, where the CNF encoding has less than 25000 clauses and a treewidth upper-bound less than 200. For each instance we round the number of clauses down to the next multiple of 1000 and round the treewidth upper-bound down to the next multiple of 10. We group the instances with the same (rounded) values and add one data point to the plot. Here, the size corresponds to the number of instances in the group and the color corresponds to the average runtime of the instances in the group.

varies significantly between $T_{\mathcal{P}}$-Unfolding and JN(.):

- The resulting CNFs from JN(.) often have many more clauses than those from $T_{\mathcal{P}}$-Unfolding. This matches the experimental results of Fierens, Van den Broeck, Thon, Gutmann, and De Raedt [Fie+11], who made similar observations for MJ(.) and JN(.) on small instances. However, while for MJ(.) this effect only occurs for small instances, we observe this effect more consistently for $T_{\mathcal{P}}$-Unfolding. Here, the better asymptotic guarantees of JN(.) do not take effect on the instances that can be solved after $T_{\mathcal{P}}$-Unfolding or JN(.)
- The treewidth upper-bound is often lower for JN(.) than for $T_{\mathcal{P}}$-Unfolding. We thus suspect that while the constant factors in the size and treewidth guarantee that comes with $T_{\mathcal{P}}$-Unfolding are both small, only the constant factor for the treewidth guarantee that comes with JN(.) is small.
- For low treewidth upper-bounds and low numbers of clauses the instance density is higher for $T_{\mathcal{P}}$-Unfolding than for JN(.). This is again explained by small constant factors

in the size and treewidth guarantees.

- $T_{\mathcal{P}}$-Unfolding leads to a solution for significantly more instances, namely 119 vs. 66 for JN(.). We especially highlight here that $T_{\mathcal{P}}$-Unfolding produces some CNF encodings with treewidth upper-bound above 100, which however are solved quickly (indicated by the blue color of the data points). This is not entirely surprising, since for $T_{\mathcal{P}}$-Unfolding we expected[11] the actual performance to supersede the guarantees entailed by treewidth. And, as we already mentioned, the binary counters that JN(.) uses may also have a negative impact on performance.

The results match the expectations that we had for the different cycle breakings. When MJ(.) results in a solution, then the CNF encoding is usually small. However, even in this case we may reach relatively high treewidth upper bounds and as a consequence, solve very few instances. JN(.) results in (relatively) large CNF encodings on which knowledge compilation is still feasible, presumably because it has considerably lower treewidth upper bounds than MJ(.) as guaranteed by Theorem 154. $T_{\mathcal{P}}$-Unfolding is somewhere in between: successfully evaluated instances can reach remarkable CNF sizes and still be solved. At the same time, they can have higher treewidth upper bounds than the ones coming from JN(.); combined with medium CNF sizes, this seems to be less of an issue than for the other encodings. We attribute this to the arguably lower hardness of the encoding obtained by $T_{\mathcal{P}}$-Unfolding compared to JN(.). While this results in a worse asymptotic scaling, it apparently leads to a smaller CNF size on instances that can still be solved by knowledge compilation in principle. We suspect that this in combination with the lower "semantic complexity" (i.e., lack of binary counters) is the cause for the performance improvement of $T_{\mathcal{P}}$-Unfolding compared to the other cycle breakings.

Q2.1. Variants of Clark's Completion Recall that we compare the differences in performance caused by Clark(.), PClark(.), and IClark(.), the different versions of Clark's Completion, when using c2d and miniC2D with a tree decomposition-guided variable order for knowledge compilation. The results are shown in Figure 4.11, where Clark(.), PClark(.), and IClark(.) are denoted by "-n", "-o", and "-b", respectively. We see that both for c2d and miniC2D Clark(.) is outperformed by PClark(.) and IClark(.). Especially, IClark(.) exhibits a significant performance increase compared to Clark(.).

We noticed that while tree decomposition guidance is beneficial for the overall performance, this is not so clear cut for the runtime of individual instances that Figure 4.12 shows. We see that IClark(.) mostly leads to better performance, especially for c2d but to a lesser extent also for miniC2D. However, in both cases, there are also some instances that are solved slower using IClark(.) than using Clark(.). We assume this is because IClark(.) yields more general worst case guarantees but may lead to an increase in CNF size and treewidth by a constant factor compared to Clark(.) when the high treewidth is inherent rather than due to translation issues that are addressed by tree decomposition guidance (i.e., rules with long bodies or many rules with the same head). In this case,

[11] Recall the observation after Lemma 159

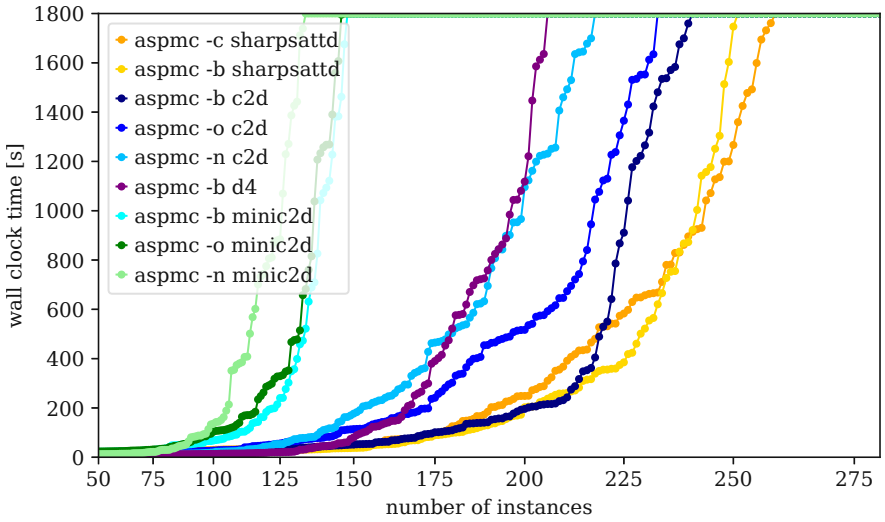


Figure 4.11: Cactus plot of the results for selected aspmc configurations over all benchmark instances, ordered (top is better) by the number of solved instances.

the performance naturally suffers a bit. This matches the intuition that we get from the scatter plots in Figure 4.12. While sometimes no guidance leads to better performance, the gain is somewhat limited.

Nevertheless, we chose to add an additional configuration to aspmc in order to test whether we can counter this phenomenon. The idea is that we only use guidance if it is expected to provide a benefit. To estimate the latter, we compute an approximate prediction for the treewidths of the CNFs obtained by all three versions of Clark's Completion and use the version with the lowest value. We included the runtime of this configuration of aspmc in Figure 4.11 together with the knowledge compiler sharpSAT-TD as "aspmc -c sharpsattd". While it has higher runtime than "aspmc -b sharpsattd" on many instances, it indeed solves a few more instances. Therefore, we chose to include it for sharpSAT-TD.

Q2.2. Effect of Knowledge Compiler Apart from Clark's Completion, also the knowledge compiler used to obtain a tractable circuit representation has an effect on the performance. The results of the comparison of such compilers are given in Figure 4.11. Here, we see that sharpSAT-TD leads to the best performance as expected; c2d and d4 have worse performance but both outperform miniC2D significantly. Interestingly, c2d solves more instances than d4 although d4 performed better in the weighted track of the 2021 model counting competition. We suspect the comparatively poor performance of miniC2D results as it produces SDDs with a fixed vtree, while all other solvers produce sd-DNNFs. This means that variables always need to occur in the same order in every branch of the produced circuit, which lowers the benefit of unit propagation in the solver.

Given the findings of *Q2.1* and *Q2.2*, we conclude that configuration "aspmc -c sharpsattd",

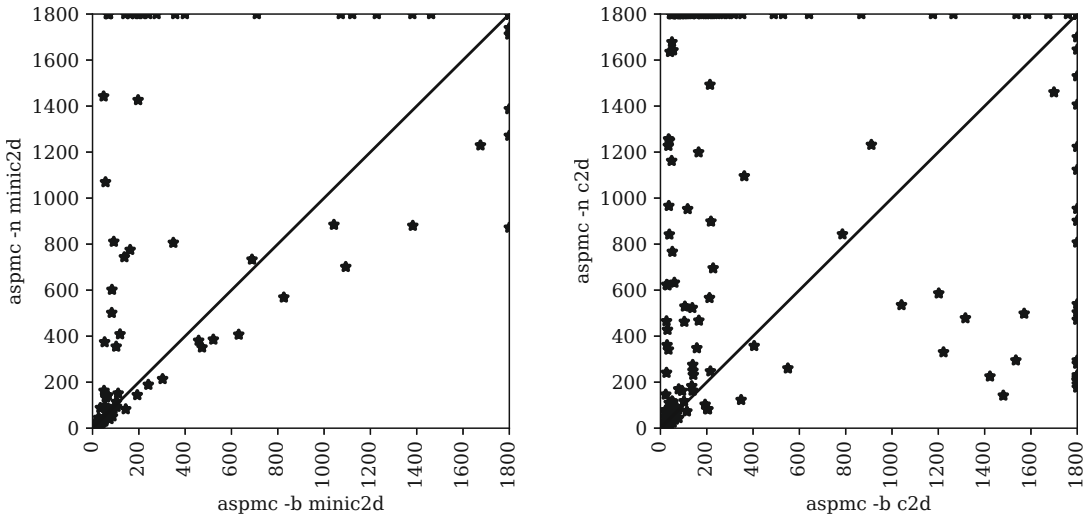


Figure 4.12: Scatter plots comparing the runtimes of individual files computed by configuration "-b" (incidence guiding), compared to "-n" (no guiding) for miniC2D (left) and c2d (right).

i.e., dynamically choosing which version of Clark's Completion to apply based on approximations of the treewidth of the resulting CNF in combination with SHARPSAT-TD as knowledge compiler, is the most promising configuration for aspmc.

Q3. Overall Performance Having established the best configuration for aspmc, we investigated the overall performance of aspmc compared to other solvers for probabilistic inference. The results are summarized in Figure 4.13.

We observe that aspmc's best configuration ("aspmc -c sharpsattd") has by far the best overall performance of all solvers with 259 solved instances. But also the other configurations, except the one that uses miniC2D performed very well, with 251 ("aspmc -b sharpsattd"), 241 ("aspmc -b c2d"), and 206 ("aspmc -b d4") solved instances. ProbLog solved 224 instances when using c2d as the knowledge compiler ("ProbLog + c2d"), however interestingly it only solved 120 and 119 instances when DSHARP ("ProbLog + Dsharp") and PySDD ("ProbLog-k sdd") are used, respectively.

The next best performing approach was LP2LP2[12], which solved 213 instances - only 7 more than "aspmc -b d4". LP2LP2 and "aspmc -b d4" are followed by PITA, which solved 154 instances, thus beating "aspmc -b minic2d", which only solved 148 instances. Last but not least, LP^{MLN} solved 24 instances.

It is unsurprising that LP^{MLN} solved the least instances, since it explicitly enumerates all the models. For probabilistic inference on ProbLog programs this quickly becomes

[12]The results for LP2LP2 must be taken with a grain of salt, since model counting over an sd-DNNF is slightly easier than evaluating multiple probabilistic queries over an sd-DNNF.

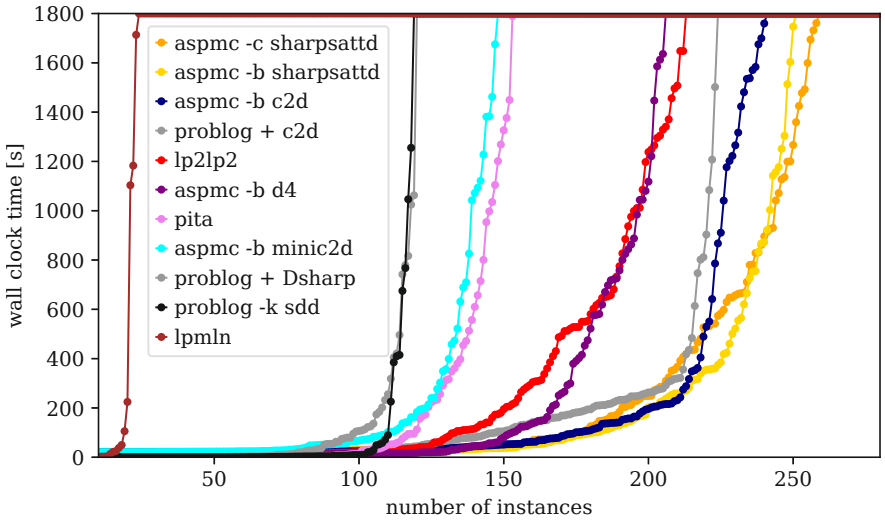


Figure 4.13: Cactus plot of the results for selected solver configurations over all benchmark instances, ordered (top is better) by the number of solved instances.

infeasible, since the number of answer sets it needs to consider is exactly 2^F, where F is the number of probabilistic facts of the program.

Similarly, it was expected that aspmc solved the most instances, due to our advancements of cycle breaking and Clark's Completion combined with the high efficiency of modern knowledge compilers.

The results of ProbLog in comparison to the other solvers are surprising. In Q1, we found that JN(.) outperforms MJ(.) on many instances but has worse performance than $T_{\mathcal{P}}$-Unfolding on the relevant instances. Thus, one may expect that the different cycle breakings determine the performance on cyclic instances and that on acyclic instances the performance mostly depends on the knowledge compiler. Here, we however see that this is not the case. While aspmc solves more instances than LP2LP2 and ProbLog, LP2LP2 solves less instances overall, even though it uses the likely best knowledge compiler (i.e. SHARPSAT-TD) and solves more cyclic instances.

Additionally, using c2d instead of DSHARP for compilation in ProbLog seems to have a huge positive impact on the performance; while a positive effect was not completely unexpected, the extent was surprising.

The relation of the performance of ProbLog's compilation to SDDs ("ProbLog -k sdd") and PITA, however, may come as a surprise. The latter uses bottom up compilation to BDDs, whereas the former performs bottom up compilation to SDDs, which have strictly better theoretical guarantees than BDDs. However, BDDs may already be good enough; in this case a more complex representation using SDDs may come with unnecessary overhead. Furthermore, neither implementation uses the theoretical guarantees explicitly by computing a vtree/variable order e.g. from a tree decomposition. Instead, PITA's

BDD library CUDD [Som12] uses dynamic variable reordering to speed up compilation. While PySDD also has a similar feature, its usage has been disabled in ProbLog for probabilistic inference.[13] The fact that ProbLog does not solve any of the blood instances, which have a simple structure that could be exploited, using option "-k sdd" but solves some (resp. all of them) when compiling to sd-DNNF with DSHARP (resp. c2d), as can be observed in Table 4.1. Apart from that, PITA does not use the same idea for handling cyclic dependencies as ProbLog, which may also contribute to the improved performance on cyclic instances.

This covers the overall performance comparison of the different solvers. We consider in more detail the performance on the different benchmark sets, shown in Table 4.1. Mostly, the results per benchmark set are reflected by the general performance already. However, we want to highlight some interesting findings separately.

Especially an explanation of the high overall performance of "ProbLog + c2d" is of interest here, given the relatively poor performance on cyclic instances. We see immediately that the high performance comes from the blood and gnb sets. Not only does "ProbLog + c2d" solve the most instances here, it is also noticeably faster than other configurations. While it also solves the most instances on the gh set, we see that there is a tie with aspmc here.

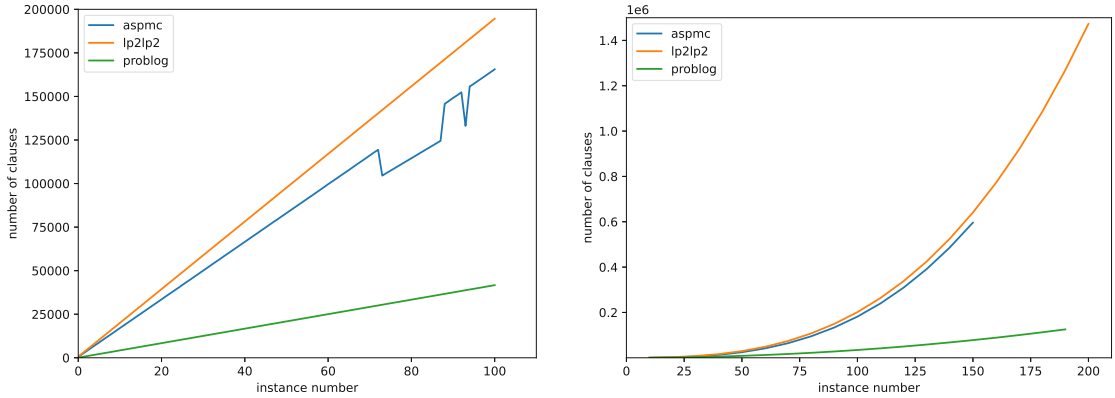
One may think that merely the use of c2d as a compiler causes the performance. However, on the cyclic benchmark sets "ProbLog + Dsharp" actually solves slightly more instances than "ProbLog + c2d". Additionally, we see that also "aspmc-b c2d" did not solve as many instances on the blood and gnb data sets as "ProbLog + c2d". Hence, there is likely an additional different reason for the performance.

We investigated ProbLog more closely and found that while its Clark Completion is standard (i.e. corresponding to Clark(.)), it makes use of a manifold of preprocessing techniques on the program level, in order to reduce the size of the program representation before Clark's Completion is applied [Sht15; Fie+15]. We therefore compared the CNF sizes produced by ProbLog, LP2LP2, and aspmc to see if there were significant differences on the gnb and blood sets.

The results are given in Section 4.8.3. We indeed see a significant difference in the sizes of the CNFs produced by the different tools. While aspmc and LP2LP2 behave similarly, ProbLog's encodings are much smaller due to its program level preprocessing capabilities. The smaller size and the usage of c2d as a knowledge compiler indeed reasonably explain why ProbLog is so much faster on these instances. Especially the blood instances have low treewidth and thus the performance bottle neck is not the complicated structure (as it is e.g. for the smokers instances) but the size of the encoding.

Apart from that, let us closely look at using aspmc with d4. We note that even though the performance is not extremely high in general, it performs best on the benchmark sets gnb and tree. We attribute this to the fact that d4 uses a different heuristic for the

[13] At least this is what we found when inspecting the source code of ProbLog.



(a) (b)

Figure 4.14: Numbers of clauses of the CNF produced by aspmc, lp2lp2 and ProbLog on (a) the blood benchmarks and (b) the gnb benchmarks.

selection of variables during compilation than those of the other compilers, which are all primarily tree decomposition guided. However, while this seems to help for gnb and tree, on the blood benchmark set a tree decomposition guided heuristic seems much better. Here, d4 only solves 29 instances, whereas aspmc with sharpSAT-TD solves more than 70. This makes sense, since the maximum treewidth upper-bound for blood was only 10, which is rather low for an acyclic instance.

Next, we take a closer look at the cyclic instances, i.e., the smokers and tree benchmark sets. Here, PITA turns out to be the best non-aspmc based approach, solving 74 instances overall. While "aspmc -c sharpsattd" solves significantly more instances, namely 119, this is still better than lp2lp2, which solved 66 instances even though compilation is performed with sharpSAT-TD. Also ProbLog only solved 55, 31, and 32 instances, in its configurations "-k sdd", "+ Dsharp", and "+ c2d", respectively.

Summing up, we observe that on all benchmark sets except for blood a configuration of aspmc performed best or second best. Second, we saw that ProbLog performs very well in the acyclic case. However, this is not due to a better strategy for knowledge compilation or Clark's Completion but due to its advanced preprocessing capabilities. Considering the preprocessing techniques of ProbLog for aspmc is thus interesting and desirable, which which however is non-trivial and goes beyond the scope of this work.

Nevertheless, this supports the claim that combining our significant improvements for cycle breaking, Clark's Completion, and the knowledge compilation step in one solver together with ProbLog's preprocessing techniques will result in even better performance on both cyclic and acyclic instances. Thus, this work brings us significantly closer to a scaleable approach for probabilistic inference.

Table 4.1: Results of the overall comparison of the different solvers and configurations. The columns show the solver, the sum of solved instances on the benchmark set, the maximum treewidth (upper bound) of a solved instance, followed by the number of solved instances in different ranges of treewidth upper bounds. The last two columns show the number of instances solved by this solver only and the total time spent on the benchmark set. Boldface highlights best values.

		tw range group					
solver	$\sum$	max(tw)	0-10	10-20	>20	unique	time[h]
growing negated body							
problog + c2d	**64**	**191**	**9**	**10**	**45**	**5**	**2.15**
aspmc -b d4	59	141	**9**	**10**	40	0	4.33
aspmc -c sharpsattd	59	141	**9**	**10**	40	0	4.88
aspmc -b sharpsattd	58	131	**9**	**10**	39	0	4.92
lp2lp2	56	121	**9**	**10**	37	0	5.66
problog + Dsharp	53	81	**9**	**10**	34	0	7.09
problog -k sdd	52	71	**9**	**10**	33	0	6.68
aspmc -b c2d	51	61	**9**	**10**	32	0	8.54
pita	50	61	**9**	**10**	31	0	10.42
aspmc -b minic2d	24	25	**9**	**10**	5	0	21.18
lpmln	6	7	6	0	0	0	29.51
blood							
problog + c2d	**100**	**10**	**100**	**0**	**0**	**13**	**4.27**
lp2lp2	75	**10**	75	**0**	**0**	0	22.11
aspmc -c sharpsattd	73	**10**	73	**0**	**0**	0	23.48
aspmc -b c2d	68	**10**	68	**0**	**0**	0	22.34
aspmc -b sharpsattd	67	**10**	67	**0**	**0**	0	21.18
aspmc -b d4	29	**10**	29	**0**	**0**	0	39.56
pita	14	**10**	14	**0**	**0**	0	43.85
aspmc -b minic2d	11	**10**	11	**0**	**0**	0	46.87
problog + Dsharp	9	**10**	9	**0**	**0**	0	46.06
problog -k sdd	0	0	0	**0**	**0**	0	50.5
lpmln	0	0	0	**0**	**0**	0	50.5
near tree							
aspmc -b d4	**22**	**53**	**1**	**6**	**15**	**3**	**15.68**
aspmc -c sharpsattd	19	50	**1**	**6**	12	0	16.37
aspmc -b c2d	18	50	**1**	5	12	0	16.49
aspmc -b sharpsattd	16	50	**1**	4	11	0	17.17
aspmc -b minic2d	16	50	**1**	5	10	0	17.48
problog + Dsharp	13	50	**1**	4	8	0	18.67
problog + c2d	13	50	**1**	4	8	0	18.96

Continued on next page

Table 4.1: Detailed results of the overall comparison. (Continued)

		tw range group					
solver	∑	max(tw)	0-10	10-20	>20	unique	time[h]
pita	10	36	**1**	5	4	0	20.62
lp2lp2	9	29	**1**	5	3	0	21.63
problog -k sdd	3	13	**1**	2	0	0	23.52
lpmln	0	0	0	0	0	0	25.0
smoker							
aspmc -b sharpsattd	**92**	**7**	**92**	**0**	**0**	**3**	131.76
aspmc -c sharpsattd	90	**7**	90	**0**	**0**	2	**131.3**
aspmc -b c2d	88	**7**	88	**0**	**0**	1	132.84
aspmc -b minic2d	82	**7**	82	**0**	**0**	0	136.44
aspmc -b d4	81	**7**	81	**0**	**0**	0	135.97
pita	64	6	64	**0**	**0**	0	144.34
lp2lp2	57	6	57	**0**	**0**	0	147.89
problog -k sdd	52	6	52	**0**	**0**	0	149.38
problog + Dsharp	29	4	29	**0**	**0**	0	160.45
problog + c2d	28	4	28	**0**	**0**	0	160.59
lpmln	12	3	12	**0**	**0**	0	168.75
growing head							
problog + c2d	**19**	**21**	8	**10**	**1**	**1**	**3.54**
aspmc -c sharpsattd	18	**21**	**9**	8	**1**	0	4.18
aspmc -b sharpsattd	18	19	**9**	9	0	0	4.47
pita	16	18	8	8	0	0	4.56
lp2lp2	16	17	**9**	7	0	0	4.72
problog + Dsharp	16	18	8	8	0	0	4.75
aspmc -b c2d	16	17	**9**	7	0	0	4.97
aspmc -b minic2d	15	16	**9**	6	0	0	5.15
aspmc -b d4	15	16	**9**	6	0	0	5.16
problog -k sdd	12	14	8	4	0	0	6.53
lpmln	6	8	6	0	0	0	9.98
Σ							
aspmc -c sharpsattd	**259**	141	**182**	**24**	53	2	180.22
aspmc -b sharpsattd	251	131	178	23	50	3	**179.51**
aspmc -b c2d	241	61	175	22	44	1	185.18
problog + c2d	224	**191**	146	**24**	54	**19**	189.51
lp2lp2	213	121	151	22	40	0	202.01
aspmc -b d4	206	141	129	22	**55**	3	200.71
pita	154	61	96	23	35	0	223.8
aspmc -b minic2d	148	50	112	21	15	0	227.13

Continued on next page

Table 4.1: Detailed results of the overall comparison. (Continued)

		tw range group					
solver	$\sum$	max(tw)	0-10	10-20	>20	unique	time[h]
problog + Dsharp	120	81	56	22	42	0	237.03
problog -k sdd	119	71	70	16	33	0	236.6
lpmln	24	8	24	0	0	0	283.75

4.9 Discussion

We first provide an overall summary and then we turn to issues for future work.

4.9.1 Summary & Findings

We considered the knowledge compilation step that is usually used in order to solve Algebraic Answer Set Counting (AASC) problems, revealing interesting new insights. Its incorporation into the AASC solver aspmc led to significant performance improvements, especially on instances with positive cyclic dependencies.

First, we considered a variety of different approaches to the knowledge compilation step. For the target representation MODS exploited in enumeration-based approaches, we consider each answer set exactly once, resulting in high efficiency for "few" answer sets but insurmountable intractability otherwise. Interestingly, most of the currently available worst case guarantees for the other, more succinct tractable circuit representations are based on the same kind of structural parameters. For compilation to sd-DNNFs and SDDs, we can derive from Theorems 137, 139 and 149 upper bounds in terms of the treewidth of a Boolean circuit representing the program. This applies regardless of whether compilation proceeds in a bottom up (using the apply operator for SDDs) or top down manner (using a knowledge compiler for CNFs). Similarly to SDDs, BDDs have performance guarantees in terms of the pathwidth of a Boolean circuit representing the program, for both bottom up and top down compilation.

There exist, however, vast differences in the quality of the guarantees. These different guarantees favor top down compilation to sd-DNNF unless the given program is much larger than the number of inputs to a Boolean circuit representing the program. Apart from that, the theoretical guarantees for SDDs seem to be better than those for BDDs, especially since every BDD can be represented as an SDD. Arguably, SDDs also come at an overhead incurred by their potentially smaller sizes. Nevertheless, one would expect SDDs to perform better than BDDs for bottom up compilation.

Second, given our preference for top down compilation to sd-DNNFs, we investigated how programs can be favorably translated to CNFs for compilation afterwards. For the customary steps in such a translation, i.e., cycle breaking and Clark Completion, we provided advancements.

For Clark's Completion, Hecher [Hec22]'s idea of guiding it with a tree decomposition in PClark(.) limits the increase of the treewidth to a factor of three compared to the *primal* treewidth of the program. We instead provided the same result for the *incidence* treewidth of the program using IClark(.), which may result in even better guarantees.

For cycle breaking, we analyzed the approaches MJ(.) [MJ10] and JN(.) [JN11], providing novel result showing upper bounds on the treewidth increase incurred by both translations. Here, we found guarantees for MJ(.) that are rather weak: Theorem 152 only gives a guarantee that grows linear in the number of simple positive cycles in the program and the number of atoms that are in positive cyclic dependencies. While this upper bound may not be tight, the lower bounds of Theorem 153 show that there is a family $(\Pi_n)_{n\in\mathbb{N}}$ of programs with constant treewidth and exactly one simple positive cycle such that the treewidth of $(\mathrm{MJ}(\Pi_n))_{n\in\mathbb{N}}$ grows at least linearly in the size of the program. The guarantees of Theorem 154 for JN(.) on the other hand are rather strong, leading only to an increase by at most a logarithmic factor in the size of the largest SCC of the positive dependency graph $\mathrm{DEP}(\Pi)$ of the program Π. Nevertheless, previous experimental results revealed that large constant factors in the size bound for JN(.) limit the effectiveness of JN(.) for probabilistic reasoning on instances that are not large enough for the asymptotic guarantees to kick in [Fie+11]. We therefore introduced a new cycle breaking called $\mathrm{T}_{\mathcal{P}}$-Unfolding. While the latter may lead to a linear increase of the treewidth in the size of the largest SCC of $\mathrm{DEP}(\Pi)$ this may be avoided in many cases, unless $\mathrm{DEP}(\Pi)$ is highly dense. To achieve this, we introduced the component-boosted backdoor size cbs(.), which intuitively measures the distance to directed or undirected acyclicity, similar to parameters such as elimination distance [GHN04]. We used cbs(.) in Theorem 155 to bound both the size increase of the program and treewidth, notably with low constant factors.

The implementation of the theoretical advancements in the AASC solver aspmc proved to be successful. The in-depth experimental evaluation confirmed our hypotheses on the effectiveness of the different approaches to cycle breakings, showing that while $\mathrm{T}_{\mathcal{P}}$-Unfolding can lead to higher treewidth upper bounds than JN(.), the latter together with the lower CNF size and presumably lower "semantic complexity" are still sufficient to allow for improved efficiency in the compilation step. Overall we see that in the cyclic setting, MJ(.), JN(.), and $\mathrm{T}_{\mathcal{P}}$-Unfolding solved 44, 66, and 119 instances, respectively, when cycle breaking is followed by CNF conversion and compilation to sd-DNNF.

Also investigating Clark's Completion paid off: both c2d and miniC2D solve the largest number of instances, if we use the incidence guided version of Clark's Completion.

Last but not least, we see most of our assumptions on the relationship between the various knowledge compilation techniques confirmed. That is, the compilation of CNFs to sd-DNNFs provides very good performance, as long as the MJ(.) cycle breaking is avoided. Unsurprisingly, the enumeration-based approach of $\mathrm{LP}^{\mathrm{MLN}}$ struggles on the probabilistic inference benchmark sets that we considered: they always have exponentially many answer sets in the number of probabilistic facts. Previous work however showed that for programs that have a complicated structure but not many answer sets, enumeration can

be beneficial [EHK21]. While different explanations seem possible, the exact cause of the low performance of bottom up compilation to SDD compared to bottom up compilation to BDDs remains somewhat mysterious.

Summing up, our work not only improves the performance of probabilistic inference, and as such AASC evaluation beyond that, but also sheds light on why some approaches work better than others based on a solid theoretical analysis.

4.9.2 Outlook

While our theoretical considerations of the different knowledge compilation approaches give some intuition on when and why they should be successful, the non-aligning empirical results make a further investigation desirable. In particular, the fact that PITA's bottom up compilation to BDDs outperforms ProbLog's bottom up compilation to SDDs. Possible differences in the way positive cyclic dependencies are handled may explain some of the differences but not the poor performance of this variant of ProbLog on the blood benchmark set. That ProbLog's implementation uses a randomly generated vtree and no dynamic variable ordering, whereas the BDD implementation of PITA employs dynamic variable reordering may be a possible factor of the performance difference. It seems plausible that dynamic variable reordering for SDDs is more expensive than for BDDs and thus disabled. However, it would be interesting to explore possibilities to generate a vtree using the structure of a Boolean circuit representing the program. This seems possible based on a tree decomposition of the Boolean circuit, given results such as Theorem 139. However, for bottom up compilation we are unaware of work in this direction.

Apart from this, preprocessing on the program level before cycle breaking is an interesting yet unexplored topic in the context of general AASC. The only simplifications aspmc currently uses are those that occur while clingo grounds the instance. In contrast, ProbLog and PITA employ a wide range of simplifications: identifying relevant ground parts of the program [Fie+15], program level size optimizations [Sht15], and using zero/one probabilities [RS11]. As we have seen, they can have a significant impact on evaluation.

While aspmc still solved the most instances overall, we can expect another significant performance boost by combining such preprocessing techniques and the advancements of aspmc. However, these simplifications are geared towards ProbLog programs and not all of them trivially generalize to AASC setting, where a more general class of programs (e.g., non-stratified ones) is allowed. It would be interesting to adopt them to the AASC setting, and possible find new techniques, especially since preprocessing in the context of model counting can have significant benefits [LM14].

An open question is also whether the double exponential upper bound of answer set counting in terms of treewidth k, proven in [JPW09], is tight when using treewidth as the only parameter. For checking whether some answer set exists, we know a $2^{\mathcal{O}(k \log(k))} \cdot |\Pi|$ upper bound that cannot be substantially improved under the Strong Exponential Time Hypothesis [Hec22], but for answer set counting the gap remains.

Besides the aforementioned points that focus on AASC specifically, there are two open problems relevant in a bigger picture. First, many preprocessing techniques on CNFs that contribute to the success of propositional model counters do not trivially generalize to the weighted/algebraic setting. It would be interesting to see whether and how we can achieve an efficient such generalization. Second, it would be interesting to see whether compilation of acyclic programs can deliver faster knowledge compilation given low incidence treewidth. Our technique using the incidence-guided version of Clark's Completion guarantee an upper bound of $\mathcal{O}(2^{3k} \cdot |\Pi|)$ on the time needed for model counting; however, recent work showed that model counting on a CNF C parameterized by incidence treewidth is feasible in $\mathcal{O}(2^k \cdot poly(|C|))$ worst case time [SS20].

CHAPTER 5

Conclusion

We conclude our work by discussing how far the goals set initially in the introduction regarding a general framework, theoretical analysis, and empirical evaluation were achieved in their respective chapters. Additionally, we point out some open issues and possible future directions.

5.1 Conclusion

Algebraic LARS as a General Framework In order to obtain a general framework that combines stream reasoning with quantitative reasoning both over the set of models as well as during the construction of models, we introduced Algebraic LARS.

Algebraic LARS intuitively corresponds to the integration of following three extensions of ASP:

(i) Algebraic Measures,

(ii) Algebraic Constraints, and

(iii) LARS.

Here, (i) allows us to perform reasoning over the set of models of a logical theory by employing the abstract algebraic structure of semirings to parameterize quantitative reasoning in a form of Weighted Logic. We saw that algebraic measures are not only general enough, to capture a wide range of other extensions, e.g., for probabilistic reasoning and preferential reasoning, but that we can faithfully incorporate the semantics of previous works. That is, we can interpret programs of previous extensions by translation to algebraic measures and obtain a conservative extension of the semantics.

We observed a similar effect for (ii), which allows us to integrate intricate non-ground constraints over quantities in the specification of programs. For this, we again successfully made use of semirings to parameterize the numerical computation. Apart from that, we employed a weighted version of Here-and-There Logic to make sure that the semantics of algebraic constraints can match the non-monotonicity that is present in other extensions in this direction. Also here, we showed that many of the previous constructs for quantitative constraints can be easily embedded in a conservative manner into our formalism.

Finally, to integrate (iii), the stream reasoning framework by Beck, Dao-Tran, and Eiter [BDE18], and (i) and (ii) properly, we had to ensure that their capabilities carry over to the temporal domain. Due to the definition of both algebraic measures and algebraic constraints in terms of Weighted (Here-and-There) Logic over semirings, this proved to be rather easy even though lengthy task. We thus consider Algebraic LARS to be a fully fledged framework for general quantitative stream reasoning with appropriate semantics that has all the desiderata stated in the introduction.

Theoretical Results Apart from the introduction of the different algebraic frameworks, we also analyzed their theoretical properties. Here, we proved a variety of different result, including non-trivial safety conditions for the variables in algebraic constraints, restrictions to ensure that programs with algebraic constraints are finitely ground, as well as the characterization of Weighted Automata in terms of a fragment of Algebraic LARS.

Mostly however, we focused on the complexity of different reasoning tasks in the context of fragments of Algebraic LARS. We found that even restricted to ASP, i.e., without temporal reasoning, algebraic constraints can easily introduce undecidability for model existence and strong equivalence. At least model checking stays feasible in PSPACE given reasonable restrictions on the used semirings. Surprisingly, in the propositional case algebraic constraints do not make any of the aforementioned tasks harder than for disjunctive ASP. We also considered the complexity of different types of queries with Algebraic LARS measures and found that unsurprisingly there is a strong connection to counting problems.

The connection to counting problems was especially interesting to study depending on the semiring parameter. As the first such work, we started an in depth consideration of the complexity of counting over semirings. Leading to a family of novel complexity classes $\mathrm{NP}(\mathcal{R})$ that characterize the complexity exactly. Interestingly, the power of $\mathrm{NP}(\mathcal{R})$ seems to depend on the chosen semiring. We showed different flavors of this connection through a series of results that connect $\mathrm{NP}(\mathcal{R})$ to well-known classical computational complexity classes. Notably, these results are not only relevant for the evaluation of algebraic measures and algebraic constraints but also for many other semiring frameworks, which we show to be $\mathrm{NP}(\mathcal{R})$-complete or $\mathrm{NP}(\mathcal{R})$-hard.

Efficient Implementation in aspmc Even though already a prototypical implementation of Algebraic LARS would have been interesting to play around with, we decided to instead focus on one particular aspect, namely algebraic measure evaluation for ASP. We

made this choice since it allowed us to invest enough time to provide aspmc, an efficient well-thought-out implementation that significantly outperforms the current state of the art solvers in probabilistic reasoning in the cyclic benchmark setting.

In order to develop aspmc, we dove deep into the fields of knowledge compilation and Fixed Parameter Tractability (FPT). Here, we analyzed different strategies for the knowledge compilation step that is usually performed in order to evaluate algebraic measures. Based on results from FPT, we found that the most promising approach is likely the translation of programs into propositional formulas, followed by knowledge compilation to the sd-DNNF tractable circuit representation. Additionally, this strategy is promising since there had been recent improvements in the performance of knowledge compilation tools in this setting.

Due to our assumption that is beneficial to first translate programs into propositional formulas, we then considered the properties of different existing translations such that we can use that translation which can be compiled the easiest. For both typical steps in such a translation, i.e., cycle breaking and Clark's Completion, we engineered improved algorithms geared towards efficient knowledge compilation.

Our implementation of these advanced methods, together with the efficiency of knowledge compilation tools for propositional formulas in aspmc, allow us to solve significantly more cyclic instances from probabilistic logic programming, with at least similar if not better performance on the non-cyclic ones.

5.2 Open Issues

Arguably, the biggest issue with Algebraic LARS is its high complexity. Already the evaluation of algebraic measures in the context of ASP has troubles scaling to big instances unless they are very well structured. Thus, an analysis of tractable fragments would be of high interest here. Apart from this, we only implemented a small part of the whole power of Algebraic LARS. It would be interesting to see how the whole framework, or at least other fragments such as LARS or even just ASP with algebraic constraints could be implemented efficiently.

Another implementation related question is how the efficiency of algebraic measure evaluation could further be improved. Here, a promising angle are preprocessing methods that simplify the program structure of the algebraic measure. While current implementations already have such techniques, they are usually restricted to probabilistic logic programs instead of the whole algebraic setting. Additionally, there are many techniques from model counting for propositional formulas that might be transferable, even though not trivially.

Also the theoretical side of counting over semirings still left some open questions. These include the question of the complexity of higher level counting problems over multiple semirings, the matter the descriptive complexity in the semiring setting, and the possibility of obtaining additional stronger results given more restrictions on the semiring.

Connecting practice and theory, a proper usage of the theoretical guarantees, based on the properties of a given semiring, in practice is still lacking. While our theoretical results hint at the fact that this should be possible, it is not clear how well this can work in practice and how current solvers for existing problems should be integrated.

While Algebraic LARS is a satisfactory framework for quantitative stream reasoning, using LARS as the basis for temporal reasoning was not the only option. Considering other options such as Temporal or Metric Equilibrium Logic would be interesting to explore.

APPENDIX A

Full Proofs: General Quantitative Stream Reasoning

A.1 Encoding Provenance of Non-ground Positive Datalog Programs

↪ Theorem 26

For provenance, we define a translation of a positive datalog program $\Pi = \{r_1, \ldots, r_m\}$ as follows. First, we discuss terminology. For each predicate q in Π we introduce the following predicates.

- $p_q(\overline{Y}, V, L, i, \overline{Z})$, which stores the value V of the provenance of $q(\overline{Y})$ using any derivation that uses exactly L leaf nodes, uses the rule r_i last and the global variables in r_i that do not occur in the head of r_i had the value $\overline{Z}$.
- $p_q(\overline{Y}, V, L, i)$, which stores the value V of the provenance of $q(\overline{Y})$ using any derivation that uses exactly L leaf nodes, uses the rule r_i last and the global variables in r_i that do not occur in the head of r_i took any value.
- $p_q(\overline{Y}, V, L)$, which stores the value V of the provenance of $q(\overline{Y})$ using any derivation that uses exactly L leaf nodes and uses any rule r_i last.
- $p_q(\overline{Y}, V)$, which stores the value V of the provenance of $q(\overline{Y})$.
- $d_q(\overline{Y}, L, i, \overline{Z})$, which asserts that there is a derivation of $q(\overline{Y})$ using exactly L leaf nodes that uses the rule r_i last and the global variables in r_i that do not occur in the head of r_i had the value $\overline{Z}$.

- $d_q(\overline{Y}, L, i)$, which asserts that there is a derivation of $q(\overline{Y})$ using exactly L leaf nodes that uses the rule r_i last and the global variables in r_i that do not occur in the head of r_i took any value.
- $d_q(\overline{Y}, L)$, which asserts that there is a derivation of $q(\overline{Y})$ using exactly L leaf nodes that uses any rule r_i.

Let

$$r_i = r(\overline{Y}) \leftarrow q_1(\overline{X}_1), \ldots, q_n(\overline{X}_n)$$

be some rule with index i, where w.l.o.g. $n > 1$ (we can always add a new extensional atom $e()$ with provenance $e_\otimes$. We add the following rules to our translation $T(\Pi)$:

$$p_r(\overline{Y}, V, L, i, \overline{Z}) \leftarrow p_{q_1}(\overline{X}_1, V_1, L_1), \ldots, p_{q_n}(\overline{X}_n, V_n, L_n), L =_{\mathbb{N}} L_1 + \ldots + L_n, \tag{A.1}$$

$$V =_{\mathcal{R}} V_1 * \ldots * V_n \tag{A.2}$$

$$d_r(\overline{Y}, L, i, \overline{Z}) \leftarrow p_{q_1}(\overline{X}_1, V_1, L_1), \ldots, p_{q_n}(\overline{X}_n, V_n, L_n), L =_{\mathbb{N}} L_1 + \ldots + L_n \tag{A.3}$$

$$p_r(\overline{Y}, V, L, i) \leftarrow d_r(\overline{Y}, L, i, \overline{Z}), V =_{\mathcal{R}} p_r(\overline{Y}, V^*, L, i, \overline{Z^*}) * V^* \tag{A.4}$$

$$d_r(\overline{Y}, L, i) \leftarrow d_r(\overline{Y}, L, i, \overline{Z}) \tag{A.5}$$

Here $\overline{Z} = \bigcup_{i=1}^{n} \overline{X}_i \setminus \overline{Y}$, i.e. the global variables of r_i that do not occur in its head.

Further, for every predicate q in Π we add the rules

$$p_r(\overline{Y}, V, L) \leftarrow d_r(\overline{Y}, L, I), V =_{\mathcal{R}} p_r(\overline{Y}, V^*, L, I^*) * V^* \tag{A.6}$$

$$d_r(\overline{Y}, L) \leftarrow d_r(\overline{Y}, L, I) \tag{A.7}$$

$$p_r(\overline{Y}, V) \leftarrow d_r(\overline{Y}, L), V =_{\mathcal{R}} p_r(\overline{Y}, V^*, L^*) * V^* \tag{A.8}$$

This translation works as follows. The last rule sums up the value of each derivation tree to obtain the final provenance, whereas the previous rules calculate the provenance of less and less restricted derivation trees. Therefore, there is at least one answer set $\mathcal{I}$ such that $p_r(\overline{Y}, V) \in \mathcal{I}$ iff the provenance of $r(\overline{Y})$ is V. On the other hand, since these rules are also all positive, there is exactly one answer set. The following results shows the correctness of the translation.

Theorem 26 (Provenance Encoding)**.** *Given a positive datalog program Π there is an $\mathcal{AC}$-program $T(\Pi)$ that computes the provenance semantics over the ω-continuous semiring $\mathcal{R}$ in the following sense. Let D be an edb and $r(\overline{x})$ a query result of $D \cup \Pi$ with semiring provenance v. Then the unique equilibrium model $\mathcal{I}$ of $T(\Pi) \cup \{p_e(\overline{x}, v, 1) \leftarrow | \ (e(\overline{x}), v) \in D\}$ contains $p_r(\overline{x}, v')$ iff $v' = v$. Where the predicates of the form $p_s(\overline{x}, v, \overline{y})$ correspond to original predicates s, with semiring label v and potential auxiliary parameters $\overline{y}$.*

Proof. Let $\mathcal{I}$ be the unique equilibrium model of $T(\Pi) \cup \{p_e(\overline{x}, v, 1) \leftarrow | \ (e(\overline{x}), v) \in D\}$. We proceed by induction on the number L of leaf nodes that are used in the derivation tree, to show that our construction is correct and the predicates indeed behave as

they should according to their description, which among other things implies that v is the provenance of the query result $r(\overline{x})$ iff the unique equilibrium model $\mathcal{I}$ of $T(\Pi) \cup \{p_e(\overline{x}, v, 1) \leftarrow | \ (e(\overline{x}), v) \in D\}$ contains $p_r(\overline{x}, v)$. In the proof we only consider the predicates for the values of the provenance in detail. The correctness of the derivability predicates $d_r(\overline{Y}, L[, i, \overline{Z}])$ follows analogous reasoning since the rules for derivability are just simplified versions of the rules for the provenance values.

The case $L = 0$ is impossible, since we always use at least one leaf node in each derivation.

The case $L = 1$ occurs exactly when $r(\overline{Y})$ is a leaf node. Since edb predicates do not occur in heads of rules in Π, the only rules we have to consider are of the form $p_e(\overline{x}, v, 1) \leftarrow$. Here, the claim holds.

Assume the claim holds for all $L' < L$.

Consider the rule (A.2). For the body to be satisfied,we need that $p_{q_1}(\overline{X}_1, V_1, L_1), \ldots, p_{q_n}(\overline{X}_n, V_n, L_n)$ are contained in $\mathcal{I}$. Since $L_i > 0$, $L = L_1 + \cdots + L_n$ and $n > 1$, we know that $L_i < L$ and therefore the claim holds for $p_{q_1}(\overline{X}_1, V_1, L_1), \ldots, p_{q_n}(\overline{X}_n, V_n, L_n)$. Therefore, $p_{q_i}(\overline{x}_i, v_i, l_i) \in \mathcal{I}$ iff v_i is the provenance of $q_i(\overline{x}_i)$ using any derivation that uses exactly l_i leaf nodes. Let us denote by $dtree_\ell(q_i(\overline{x}_i))$ the set of all derivation trees τ for $q_i(\overline{x}_i)$ that use exactly ℓ leaf nodes, and by $leaves(\tau)$ the set of leaf nodes in the tree τ. Then

$$v_i = \bigoplus_{\tau \in dtree_{l_i}(q_i(\overline{x}_i))} \ \bigotimes_{t \in leaves(\tau)} R(t). \tag{A.9}$$

Then for $v = v_1 \otimes \cdots \otimes v_n$, and $l = l_1 + \cdots + l_n$ we have that $p_r(\overline{y}, v, l, i, \overline{z})$ is in $\mathcal{I}$ iff for $i = 1, \ldots, n$ the atoms $p_{q_i}(\overline{x}_i, v_i, l_i)$ are in $\mathcal{I}$. This however means that for $i = 1, \ldots, n$ the equation (A.9) holds. Therefore v is the value

$$v_1 \otimes \ldots \otimes v_n \ = \bigoplus_{\tau \in dtree_{l_1}(q_1(\overline{x}_1))} \ \bigotimes_{t \in leaves(\tau)} R(t) \ \otimes \cdots \otimes \bigoplus_{\tau \in dtree_{l_n}(q_n(\overline{x}_n))} \ \bigotimes_{t \in leaves(\tau)} R(t).$$

We use that for every combination of derivations $\tau_1, \ldots, \tau_n$ respectively for $q_1(\overline{x_1}), \ldots, q_n(\overline{x_n})$ there is a derivation of $r(\overline{y})$ using the rule r_i last, where the global variables that do not occur in the head of r_i have the value $\overline{z}$. According to the distributive law, assuming that $last(\tau)$ denotes the last rule in derivation tree τ and that $gvar(r_i)$ denotes the value of the global variables in rule r_i that do not occur in the head of r_i, we obtain that

$$v \ = \ v_1 \otimes \ldots \otimes v_n \ = \bigoplus_{\tau \in dtree_l(r(\overline{y})),\, r_i = last(\tau),\, gvar(r_i) = \overline{z}} \ \bigotimes_{t \in leaves(\tau)} R(t).$$

It follows that rule (A.2) ensures that the predicates of the form $p_r(\overline{Y}, V, L, i, \overline{Z})$ satisfy our claim (for L).

Next for rule (A.4). Here, we simply aggregate over the global variables in r_i that do not occur in the head of r_i.

The head atom $p_r(\overline{Y}, V, L, i)$ should describe the value V of the provenance of $r(\overline{Y})$ using any derivation that uses exactly L leaf nodes and uses rule r_i last. This value is given by

$$\bigoplus_{\tau \in dtree_L(r(\overline{x})),\, r_i = last(\tau)} \quad \bigotimes_{t \in leaves(\tau)} R(t),$$

which is equal to

$$\bigoplus_{\overline{z}} \quad \bigoplus_{\tau \in dtree_l(r(\overline{x})),\, r_i = last(\tau),\, gvar(r_i) = \overline{z}} \quad \bigotimes_{t \in leaves(\tau)} R(t).$$

We know that according to the previous rule (A.2), the predicate $p_r(\overline{Y}, V, L, i, \overline{Z})$ encodes exactly the inner sum. Since rule (A.4) performs the outer sum, it follows that rule (A.4) ensures that the predicates of the form $p_r(\overline{Y}, V, L, i)$ satisfy our claim (for L).

Next, the rule (A.6) aggregates over the different rules that were used last to derive $r(\overline{x})$ using l leaf nodes. The argumentation is analogous to the one for the last rule (A.4), where we aggregated over $\overline{z}$ instead of the rule index i like here.

Overall, the inductive proof of the correctness of the rules specifying the predicates $p_r(\overline{Y}, V, L[, i, \overline{Z}])$ succeeds, since all the predicates are correctly defined for L given that predicates are well defined for $L' < L$.

Last but not least we consider the last rule (A.8) which should produce the final result. According to the definition of provenance for datalog programs in [GKT07], the label v of the query result $r(\overline{x})$ is

$$v \;=\; \bigoplus_{\tau \in dtree(r(\overline{x}))} \quad \bigotimes_{t \in leaves(\tau)} R(t),$$

where $dtree(r(\overline{x}))$ is the set of all derivation trees for $r(\overline{x})$ and $R(t)$ is the provenance of the leaf t. We reformulate this equation as follows:

$$v \;=\; \bigoplus_{\tau \in dtree(r(\overline{x}))} \quad \bigotimes_{t \in leaves(\tau)} R(t) \;=\; \bigoplus_{l \geq 0} \quad \bigoplus_{\tau \in dtree_l(r(\overline{x}))} \quad \bigotimes_{t \in leaves(\tau)} R(t);$$

since $p_r(\overline{x}, v', l) \in \mathcal{I}$ iff $v' = \bigoplus_{\tau \in dtree_l(r(\overline{x}))} \bigotimes_{t \in leaves(\tau)} R(t)$, we obtain

$$v \;=\; \bigoplus_{l \geq 0} \quad \bigoplus_{p_r(\overline{x}, v', l) \in \mathcal{I}} v'.$$

As due to rule (A.8), we have $p_r(\overline{x}, v) \in \mathcal{I}$ iff

$$v = \bigoplus_{l \geq 0} \quad \bigoplus_{p_r(\overline{x}, v', l) \in \mathcal{I}} v',$$

we obtain that $p_r(\overline{x}, v) \in \mathcal{I}$. This concludes the proof. □

A.2 Domain Independence and Safety ↪ Theorem 29 and 31

We proof Theorem 29 via the following lemma:

Lemma A.2.1 (Support Independence)**.** *Let σ_1, σ_2 be semiring signatures, $\mathcal{I}_w$ be a pointed σ_i-HT-interpretation ($i = 1, 2$) and α be a weighted σ_i-formula ($i = 1, 2$) that is syntactically domain independent w.r.t. variable x. Then*

$$\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\} = \{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\alpha(\xi)]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\}.$$

Proof. We give a proof using structural induction on the syntactically domain independent formula α. In the following let $\mathcal{I}_w$ some pointed HT-interpretation and σ_1, σ_2 semiring signatures that contain all the constants of α and $\mathcal{I}_w$

- Case $\alpha = k$:
  This formula contains no local variables, therefore the equality is trivially fulfilled.
- Case $\alpha = \phi(\overline{x})$:
  The given formulas are all range restricted. For range restricted formulas it is known, that they are domain independent (see for example [Dem92]), which implies that when they are seen as weighted formulas, their support does not depend on the signature.
- Case $\alpha = \neg_\oplus \beta(x)$:
  The semantics of $\neg_\oplus \beta$ is the inverse of the semantics of β w.r.t. $\oplus$, which is $e_\oplus$ iff the semantics of β is $e_\oplus$. Therefore we have

$$\begin{aligned}
&\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\neg_\oplus \beta(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\} \\
=&\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\beta(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\} \\
=&\{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\beta(\xi)]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\} \\
=&\{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\neg_\oplus \beta(\xi)]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\}.
\end{aligned}$$

- Case $\alpha = \neg_\otimes \beta(x)$:
  The semantics of $\neg_\otimes \beta$ is the inverse of the semantics of β w.r.t. $\otimes$ or $e_\oplus$ if the semantics of β is $e_\oplus$. Therefore we have on the one hand that

$$[\![\beta(\xi)]\!]^{\sigma_i}_{\mathcal{R}}(\mathcal{I}) = e_\oplus \Rightarrow [\![\neg_\otimes \beta(\xi)]\!]^{\sigma_i}_{\mathcal{R}}(\mathcal{I}) = e_\oplus.$$

  Furthermore, we have for the other direction that

$$[\![\neg_\otimes \beta(\xi)]\!]^{\sigma_i}_{\mathcal{R}}(\mathcal{I}) = e_\oplus \Rightarrow [\![\beta(\xi)]\!]^{\sigma_i}_{\mathcal{R}}(\mathcal{I}) = e_\oplus \vee [\![\beta(\xi)]\!]^{\sigma_i}_{\mathcal{R}}(\mathcal{I}) \otimes e_\oplus = e_\otimes$$

  The second disjunct implies that $e_\oplus = e_\otimes$ since $e_\oplus$ annihilates R. Therefore since $\forall r \in R : e_\otimes \otimes r = r$ holds we have that $\forall r \in R : e_\oplus \otimes r = e_\oplus = r$, meaning our semiring has exactly one element, namely $e_\oplus$. Therefore we have

$$[\![\neg_\otimes \beta(\xi)]\!]^{\sigma_i}_{\mathcal{R}}(\mathcal{I}) = e_\oplus \Rightarrow [\![\beta(\xi)]\!]^{\sigma_i}_{\mathcal{R}}(\mathcal{I}) = e_\oplus$$

So we know that the semantics of $\neg_{\otimes}\beta$ is $e_{\oplus}$ iff the semantics of β is $e_{\oplus}$ and as in the previous case we obtain

$$\begin{aligned}&\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\neg_{\otimes}\beta(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}\\ =&\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\beta(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}\\ =&\{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\beta(\xi)]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}\\ =&\{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\neg_{\otimes}\beta(\xi)]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}.\end{aligned}$$

- Case $\alpha = \neg\neg\alpha_1(x)$:
  We know that $\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\neg\neg\alpha_1(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}$ is equal to $\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_1(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}$. Therefore this case follows immediately from the inductive hypothesis for $\alpha_1(x)$.

- Case $\alpha = \alpha_1(x) + \alpha_2(x)$:
  Assume that there is

  $$\xi \in \mathfrak{r}_2(s(x)) \setminus \mathfrak{r}_1(s(x)) \text{ s.t. } [\![\alpha_1(\xi) + \alpha_2(\xi)]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\},$$

  then we know that

  $$[\![\alpha_1(\xi)]\!]_{\mathcal{R}}(\mathcal{I}) \neq e_{\oplus} \text{ or } [\![\alpha_2(\xi)]\!]_{\mathcal{R}}(\mathcal{I}) \neq e_{\oplus}$$

  and further that $\xi \notin \mathcal{D}_2$, since σ_1, σ_2 are semiring signatures. Then it however holds that

  $$\xi \in \{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\alpha_1(\xi)]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\} \setminus \{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_1(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}$$

  or

  $$\xi \in \{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\alpha_2(\xi)]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\} \setminus \{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_2(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}.$$

  This is impossible due to the induction hypothesis for $\alpha_1(x)$ and $\alpha_2(x)$. Analogously we can show that

  $$\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_1(\xi)+\alpha_2(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}\setminus\{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\alpha_1(\xi)+\alpha_2(\xi)]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}$$

  is empty, and therefore that

  $$\begin{aligned}&\{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\alpha_1(\xi) + \alpha_2(\xi)]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}\\ =&\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_1(\xi) + \alpha_2(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}.\end{aligned}$$

- Case $\alpha = \alpha_1(x) * \alpha_2$:
  First note that:

  $$\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_1(\xi) * \alpha_2]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\} \subseteq \{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_1(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_{\oplus}\}$$

We use the induction hypothesis on $\alpha_1(x)$ to obtain the equality:

$$\begin{aligned}
&\{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_1(\xi) * \alpha_2]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\} \\
=&\{\xi \in \{\xi \in \mathfrak{r}_1(s(x)) \mid [\![\alpha_1(\xi)]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\} \mid [\![\alpha_1(\xi) * \alpha_2]\!]^{\sigma_1}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\} \\
=&\{\xi \in \{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\alpha_1(\xi)]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\} \mid [\![\alpha_1(\xi) * \alpha_2]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\} \\
=&\{\xi \in \mathfrak{r}_2(s(x)) \mid [\![\alpha_1(\xi) * \alpha_2]\!]^{\sigma_2}_{\mathcal{R}}(\mathcal{I}_w) \neq e_\oplus\}
\end{aligned}$$

- Case $\alpha = \alpha_1 * \alpha_2(x)$:
  works analogously to the one above.
- Case $\alpha = \alpha_1(x) * \phi(\overline{X}')$, where $\overline{X}' \subseteq \{x\}$:
  works analogously to the one above.

The proof for more than one local variable works analogously. □

Theorem 29 (Formula Domain Independence)**.** *If a formula $\alpha(\overline{X})$ over semiring $\mathcal{R}$ is syntactically domain independent w.r.t. $\overline{X}$, then $\alpha^\Sigma = \Sigma\overline{X}\ \alpha(\overline{X})$ is domain independent.*

Proof. When we evaluate $\alpha(\overline{X})$ we take the sum over $\text{supp}_\oplus(\alpha(\overline{X}), \mathcal{I}_w)$. Due to the previous lemma we know that the support is invariant under changing the domain. Further, we know that for a given assignment of the local variables the semantics is independent of the domain. Therefore, the semantics is invariant under changing the domain for syntactically domain independent formulas. □

A.3 Strong Equivalence Using Finite Programs ↪ Theorem 33

Theorem 33 (Strong Equivalence)**.** *For any Π_1, Π_2 programs, $\Pi_1 \equiv_s \Pi_2$ iff Π_1 has the same HT-models, i.e. satisfying pointed HT-interpretations, as Π_2.*

Proof. We have two directions to prove. Based on the idea of [LPV01].

($\Rightarrow$) We prove this direction using contraposition, that is we assume that we have two programs Π_1, Π_2 s.t. for some HT-interpretation $(\mathcal{I}^H, \mathcal{I}^T)$ it holds that (w.l.o.g.) $(\mathcal{I}^H, \mathcal{I}^T, H) \models \Pi_1$ and $(\mathcal{I}^H, \mathcal{I}^T, H) \not\models \Pi_2$. Next we show that there exists a program Δ s.t. $\Pi_1 \cup \Delta$ and $\Pi_2 \cup \Delta$ have different answer sets.

The program Δ consists of the following rules, where $\mathcal{G}$ is the set of predicates that occur in $\Pi_1 \cup \Pi_2$:

$$\text{repair}_p(X_1, \ldots, X_n) \leftarrow \top =_{\mathbb{B}} \neg\neg\text{repair}_p(X_1, \ldots, X_n), \quad \text{for } p \in \mathcal{G} \text{ with arity } n. \tag{A.10}$$

$$p(X_1, \ldots, X_n) \leftarrow \text{repair}_p(X_1, \ldots, X_n), \tag{A.11}$$

$$\text{for } p \in \mathcal{G} \text{ with arity } n.$$

$$\text{fill}_{p,q}(X_1,\ldots,X_n,Y_1,\ldots,Y_m) \leftarrow \top =_{\mathbb{B}} \neg\neg\text{fill}_{p,q}(X_1,\ldots,X_n,Y_1,\ldots,Y_m), \tag{A.12}$$

$$\text{for p,q} \in \mathcal{G} \text{ with arities } n, m.$$

$$p(X_1,\ldots,X_n) \leftarrow q(Y_1,\ldots,Y_m), \text{fill}_{p,q}(X_1,\ldots,X_n,Y_1,\ldots,Y_m), \tag{A.13}$$

$$\text{for p,q} \in \mathcal{G} \text{ with arities } n, m.$$

Intuitively $\text{repair}_p(X_1,\ldots,X_n)$ guesses some tuple $(X_1,\ldots,X_n)$ for predicate p such that the atom $p(X_1,\ldots,X_n)$ should definitely be satisfied.

Similarly, $\text{fill}_{p,q}(X_1,\ldots,X_n,Y_1,\ldots,Y_m)$ guesses some values $(X_1,\ldots,X_n)$ and $(Y_1,\ldots,Y_m)$ for the predicates p and q, respectively, such that if $p(X_1,\ldots,X_n)$ is satisfied then also $q(Y_1,\ldots,Y_n)$ should be satisfied.

Consider now the interpretation

$$\mathcal{I}^* = \mathcal{I}^T \cup \{\text{repair}_p(x_1,\ldots,x_n) \mid p(x_1,\ldots,x_n) \in \mathcal{I}^H\} \tag{A.14}$$

$$\cup \{\text{fill}_{p,q}(x_1,\ldots,x_n,y_1,\ldots,y_m) \mid p(x_1,\ldots,x_n), q(y_1,\ldots,y_n) \in \mathcal{I}^T \setminus \mathcal{I}^H\}. \tag{A.15}$$

Then we have that $(\mathcal{I}^H,\mathcal{I}^*,H) \models \Pi_1 \cup \Delta$, therefore $\mathcal{I}^*$ is not an equilibrium model of $\Pi_1 \cup \Delta$. However for Π_2 we have that $(\mathcal{I}^*,\mathcal{I}^*,H) \models \Pi_2 \cup \Delta$. Furthermore, consider now some interpretation $\mathcal{I}' \subseteq \mathcal{I}^*$ s.t. $(\mathcal{I}',\mathcal{I}^*,H) \models \Pi_2 \cup \Delta$. Due to the included repairs, we know that at least $\mathcal{I}^H \subseteq \mathcal{I}'$. Moreover, we know that this inclusion is strict even when we consider only the predicates occurring in $\Pi_1 \cup \Pi_2$, since $(\mathcal{I}^H,\mathcal{I}^T,H) \not\models \Pi_2$. Therefore, due to the fills we have to include all the predicates from $\mathcal{I}^T$. It follows that $\mathcal{I}' = \mathcal{I}^*$ and therefore that $\mathcal{I}^*$ is an equilibrium model of $\Pi_2 \cup \Delta$.

($\Leftarrow$) Assume that Π_1 has the same HT-models as Π_2 and consider for an arbitrary program Δ the HT-models of $\Pi_1 \cup \Delta$ and $\Pi_2 \cup \Delta$. Those are exactly the HT-models $(\mathcal{I}^H,\mathcal{I}^T)$ s.t.

$$(\mathcal{I}^H,\mathcal{I}^T,H) \models \Pi_1 \text{ and } (\mathcal{I}^H,\mathcal{I}^T,H) \models \Delta,$$

which is however equivalent to

$$(\mathcal{I}^H,\mathcal{I}^T,H) \models \Pi_2 \text{ and } (\mathcal{I}^H,\mathcal{I}^T,H) \models \Delta$$

since Π_1 and Π_2 have the same HT-models. Since the HT-models of $\Pi_1 \cup \Delta$ and $\Pi_2 \cup \Delta$ are the same, we also know that the equilibrium models $\mathcal{I}$ are the same, since it follows that

$$\begin{aligned} &(\mathcal{I},\mathcal{I},H) \models \Pi_1 \cup \Delta \text{ and } \forall \mathcal{I}' \subsetneq \mathcal{I} : (\mathcal{I}',\mathcal{I},H) \not\models \Pi_1 \cup \Delta \\ \iff &(\mathcal{I},\mathcal{I},H) \models \Pi_2 \cup \Delta \text{ and } \forall \mathcal{I}' \subsetneq \mathcal{I} : (\mathcal{I}',\mathcal{I},H) \not\models \Pi_2 \cup \Delta. \end{aligned}$$

□

A.4 Complexity of Reasoning with $\mathcal{AC}$-Programs ↪ Theorem 37

We first prove, as an auxiliary lemma, that we can evaluate variable free weighted formulas over efficiently encoded semirings in polynomial time.

Lemma A.4.2 (Complexity of evaluation)**.** *Let $\mathcal{R} = (R, \oplus, \otimes, e_\oplus, e_\otimes)$ some semiring and $e : R \to \mathbb{N}$ some encoding function s.t. $\mathcal{R}$ is efficiently encoded by e.*

Then for a quantifier-free weighted formula over $\mathcal{R}$ and pointed HT-interpretation $\mathcal{I}_w$, we can calculate $e([\![\alpha]\!]_\mathcal{R}(\mathcal{I}_w))$ in polynomial time.

Proof. The proof is by structural induction on the formula α, with induction invariant that $t(\alpha)$ the time needed is in $\mathcal{O}(N^n)$, where N is the size of the input, $n \in \mathbb{N}$ is a constant not depending on the input. Further, $s(\alpha)$ the size of the representation of the obtained value, i.e. $\|[\![\alpha]\!]_\mathcal{R}(\mathcal{I}_w)\|$, is in $\mathcal{O}(N)$.‘

- Base Cases:
  - $\alpha = e(k)$: Then one can evaluate the expression by simply returning $e(k)$. This is feasible in polynomial time. The size of the output is linear in the size of the input.
  - $\alpha = \phi$: We simply check if $\mathcal{I}_w \models \phi$ and return $e_\oplus$ or $e_\otimes$ accordingly. This is possible in polynomial time since ϕ is quantifier-free and the size of the output is also bounded by a constant.

  We have shown the invariant for all formulae up to a certain structural complexity.

- Induction Step:
  - $\alpha = \beta_1 \to \beta_2$: We know that for β_i the invariant holds, therefore we can check in time bounded polynomially in the size of the formula, whether $[\![\beta_i]\!]_\mathcal{R}(\mathcal{I}_w) = e_\oplus$ and output $e_\oplus$ or $e_\otimes$ accordingly. The size of the output is again bounded by a constant.
  - $\alpha = \beta_1 + \beta_2$: We know that the invariant holds for β_1, β_2. Further $t(\alpha) = t(\beta_1) + t(\beta_2) + x$, where x is the time needed for addition of the results for β_1 and β_2. We know that x is polynomial in $s(\beta_1) + s(\beta_2)$, which we know to be in $\mathcal{O}(N)$. Therefore $x \in \mathcal{O}(N^l)$, where l is the degree of the polynomial bounding the time needed to add two numbers. It follows that $t(\alpha) \in \mathcal{O}(N^n)$. For $s(\alpha)$ we can see that

$$\begin{aligned} s(\alpha) &= \|[\![\alpha]\!]_\mathcal{R}(\mathcal{I}_w)\| \\ &\leq \|[\![\beta_1]\!]_\mathcal{R}(\mathcal{I}_w)\| + \|[\![\beta_2]\!]_\mathcal{R}(\mathcal{I}_w)\| + C \end{aligned}$$

  And therefore $s(\alpha) \in \mathcal{O}(N)$.

- $\alpha = \beta_1 * \beta_2$: The proof works analogously to the proof for the case $\alpha = \beta_1 + \beta_2$.
- $\alpha = -\beta$: We know that the invariant holds for β. Further $t(\alpha) = t(\beta) + x$, where x is the time needed for inversion of the result for β. We know that x is polynomial in $s(\beta)$, which we know to be in $\mathcal{O}(N)$. Therefore $x \in \mathcal{O}(N^l)$, where l is the degree of the polynomial bounding the time needed to invert a number. It follows that $t(\alpha) \in \mathcal{O}(N^n)$. For $s(\alpha)$ we can see that

$$\begin{aligned} s(\alpha) &= \|[\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}_w)\| \\ &\leq \|[\![\beta]\!]_{\mathcal{R}}(\mathcal{I}_w)\| + C \end{aligned}$$

  And therefore $s(\alpha) \in \mathcal{O}(N)$.
- $\alpha = \beta^{-\mathbf{1}}$: The proof works analogously to the proof for the case $\alpha = -\beta$.

□

Theorem 37 (Ground Complexity)**.** *For variable-free programs over efficiently encoded semirings*

- *MC and (propositional) SE are co-NP-complete, and*
- *SAT is Σ_2^p-complete.*

Proof. The hardness parts are inherited from the complexity of the respective problems for disjunctive logic programs [Dan+01; Lin02]: The disjunctive logic programming rule

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_k$$

is strongly equivalent to the $\mathcal{AC}$-rule

$$1 =_{\mathbb{B}} a_1 + \ldots + a_n \leftarrow b_1, \ldots, b_m, \neg c_1, \ldots, \neg c_k.$$

The memberships follow from the possibility of applying guess and check algorithms. We only need that given $(\mathcal{I}^H, \mathcal{I}^T)$ and algebraic constraint $k \sim_{\mathcal{R}} \alpha$, we can decide in polynomial time whether $\mathcal{I}_H \models k \sim_{\mathcal{R}} \alpha$. This is possible since we know that $\mathcal{R}$ is efficiently encoded: We only need to perform polynomially many additions, multiplications and inversions which each take polynomial time as Theorem A.4.2. □

A.5 Equivalence of the Expressiveness of LARS Measures and Weighted Automata ↪ Theorems 64 and 65

Theorem 64 (Reduction of Weighted Automata to LARS Measures)**.** *Given a weighted automaton $\mathcal{A}$ over a finite alphabet A and semiring $\mathcal{R}$, there exists a LARS measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$, s.t.*

$$\forall w \in A^* : \|\mathcal{A}\|(w) = \mu(\Pi, \tau(w)),$$

where $\tau(w)$ is a translation from words w to pairs of data streams and time points, defined by

$$\tau(w) = \left(\left([0, |w|], t \mapsto \left\{ \begin{array}{ll} \{w_{t+1}\} & \text{if } t < |w| \\ \emptyset & \text{otherwise} \end{array} \right\} \right), 0 \right)$$

Proof. In the following we use the window functions next and prev, which reduce the current stream to the next and previous time point, respectively.

Let Π consist of the rules

$$\leftarrow \Diamond \left(\bigwedge_{q \in Q} \neg q \vee \bigvee_{q,q' \in Q, q \neq q'} q \wedge q' \right)$$
$$\Box \bigvee_{q \in Q} q \leftarrow$$

Further let $\alpha = \alpha_1 * \alpha_2 * \alpha_3$, where

$$\begin{aligned} \alpha_1 =& \Diamond \Sigma_{q \in Q} \boxplus^{\text{prev}} \Box \bot * q * \lambda(q), \\ \alpha_2 =& \Box \Sigma_{q,q' \in Q, a \in A} q * a * \boxplus^{\text{next}} \Diamond q' * \delta(a)_{q,q'}, \\ \alpha_3 =& \Diamond \Sigma_{q \in Q} \boxplus^{\text{next}} \Box \bot * q * \gamma(q). \end{aligned}$$

Since the data streams we consider are of the form $\tau(w)$ for a word $w \in A^*$, the program Π has exactly those streams as answer streams, s.t. at every time point it holds, that there are is exactly one $q \in Q$ and one $a \in A$, s.t. q, a are satisfied, except at the last time point of the stream, where no $a \in A$ is satisfied and exactly one $q \in Q$ is satisfied.

For such an answer stream S we write $q : T \to Q, a : T \setminus \{\max\{t \in T\}\} \to A$, the functions, that return those q's and a's.

We then get that

$$\begin{aligned} \mu(S, \tau(w)) =& [\![\alpha]\!]_{\mathcal{R}}(S, 0) \\ =& [\![\alpha_1]\!]_{\mathcal{R}}(S, 0) \otimes [\![\alpha_2]\!]_{\mathcal{R}}(S, 0) \otimes [\![\alpha_3]\!]_{\mathcal{R}}(S, 0). \end{aligned}$$

We simplify each term α_i:

$$\begin{aligned} [\![\alpha_1]\!]_{\mathcal{R}}(S, 0) =& [\![\Diamond \Sigma_{q \in Q} \boxplus^{\text{prev}} \Box \bot * q * \lambda(q)]\!]_{\mathcal{R}}(S, 0) \\ =& \bigoplus_{t \in T} \bigoplus_{q \in Q} \left\{ \begin{array}{ll} e_{\otimes} & t = 0 \\ e_{\oplus} & \text{otherwise.} \end{array} \right\} \otimes \left\{ \begin{array}{ll} e_{\otimes} & q = q(t) \\ e_{\oplus} & \text{otherwise.} \end{array} \right\} \otimes \lambda(q) \\ =& \lambda(q(0)), \\ [\![\alpha_2]\!]_{\mathcal{R}}(S, 0) =& \left[\!\!\left[\Box \Sigma_{q,q' \in Q, a \in A} q * a * \boxplus^{\text{next}} \Diamond q' * \delta(a)_{q,q'} \right]\!\!\right]_{\mathcal{R}} (S, 0) \\ =& \bigotimes_{t \in T} \bigoplus_{q,q' \in Q, a \in A} \left[\!\!\left[q * a * \boxplus^{\text{next}} \Diamond q' * \delta(a)_{q,q'} \right]\!\!\right]_{\mathcal{R}} (S, t) \end{aligned}$$

$$= \bigotimes_{t=0}^{|T|-1} \bigoplus_{q,q' \in Q, a \in A} \begin{cases} \delta(a)_{q,q'} & q = q(t), t \neq |T|-1, a = a(t), q' = q(t+1) \\ e_\oplus & \text{otherwise.} \end{cases}$$

$$= \bigotimes_{t=0}^{|T|-2} \delta(a(t))_{q(t),q(t+1)}$$

$$[\![\alpha_3]\!]_{\mathcal{R}}(S,0) = [\![\Diamond \Sigma_{q \in Q} \boxplus^{\text{next}} \Box\bot * q * \gamma(q)]\!]_{\mathcal{R}}(S,0)$$

$$= \bigoplus_{t \in T} \bigoplus_{q \in Q} [\![\boxplus^{\text{next}} \Box\bot * q * \gamma(q)]\!]_{\mathcal{R}}(S,t)$$

$$= \bigoplus_{t \in T} \bigoplus_{q \in Q} \begin{cases} e_\otimes & t = |T|-1 \\ e_\oplus & \text{otherwise.} \end{cases} \otimes \begin{cases} e_\otimes & q = q(t) \\ e_\oplus & \text{otherwise.} \end{cases} \otimes \gamma(q)$$

$$= \gamma(q(|T|-1))$$

So α_1 gives us the contribution of the initial state, α_2 the contribution of the path between the initial and final state, w.r.t. to the word a, and α_3 gives the contribution of the final state. So overall α gives us the weight of the path represented by the stream S:

$$[\![\alpha]\!]_{\mathcal{R}}(S,0) = \lambda(q(0)) \otimes \bigotimes_{t=0}^{|T|-2} \delta(a(t))_{q(t),q(t+1)} \otimes \gamma(q(|T|-1))$$

Now let $w \in A^*$. All the possible answer streams S, that are also interpretation streams of $\tau(w)_1$ are those answer streams such that $a = w$.

Then if we sum over all interpretation answer streams we get the sum over all paths

$$\mu(\Pi, \tau(w)) = \bigoplus_{S \in \mathcal{AS}(\Pi, \tau(w)_1, 0)} [\![\alpha]\!]_{\mathcal{R}}(S,0)$$

$$= \bigoplus_{S \in \mathcal{AS}(\Pi, \tau(w)_1, 0)} \lambda(q(0)) \otimes \bigotimes_{t=0}^{|T|-2} \delta(a(t))_{q(t),q(t+1)} \otimes \gamma(q(|T|-1))$$

$$= \bigoplus_{S \in \mathcal{AS}(\Pi, \tau(w)_1, 0)} \lambda(q(0)) \otimes \bigotimes_{t=0}^{|T|-2} \delta(w(t))_{q(t),q(t+1)} \otimes \gamma(q(|T|-1))$$

$$= \bigoplus_{q:[0,|w|] \to Q} \lambda(q(0)) \otimes \bigotimes_{t=0}^{|T|-2} \delta(w(t))_{q(t),q(t+1)} \otimes \gamma(q(|T|-1))$$

$$= \lambda \otimes \delta(w) \otimes \gamma$$

□

Theorem 65 (Reduction of LARS Measures to Weighted MSO)**.** *Let $\langle \Pi, \alpha, \mathcal{R} \rangle$ be a restricted LARS measure over a semiring $\mathcal{R}$. Then there exists a restricted weighted MSO formula Ψ over $\mathcal{R}$, s.t.*

$$\forall D, t : \mu(\Pi, D, t) = [\![\Psi]\!]_{\mathcal{R}}(\sigma(D,t)),$$

where for a data stream D and time point t, $\sigma(D,t) = (\sigma_1(D,t), \sigma_2(D,t))$ s.t. σ_1 is the word that the weighted MSO formula is interpreted over and σ_2 corresponds to the assignments of free variables in Ψ. That is,

$$\begin{aligned}
\sigma_1(D,t)_i =& \{a \mid a \in v(i) \cap \mathbf{HB}(\Pi)^\epsilon\}, \\
\sigma_2(D,t)(x_0) =& t && \text{for the non-quantified first-order variable } x_0, \\
\sigma_2(D,t)(A) =& \{t \in \mathbb{N} \mid a \in v(t)\} && \text{for MSO variables } A. \\
\sigma_2((T,v),t)(T) =& T && \text{for MSO variable } T.
\end{aligned}$$

We start by introducing the translation f and showing that it is faithful.

Definition A.5.3 (Translation from LARS to MSO)**.** *Let $f : LARS_w \times (W \cup \mathbb{N})^* \to MSO(R,A)$ be defined recursively by*

$$\begin{aligned}
f(k,O) =& k \\
f(a,O) =& \begin{cases} \bigvee_{s \in 2^{\mathcal{A}^\epsilon}, a \in s} P_s^{r(O)}(x_{i(O)}) & a \in \mathcal{A}^\epsilon \\ x_{i(O)} \in A^{r(O)} & \text{otherwise.} \end{cases} \\
f(\neg a,O) =& \begin{cases} \bigwedge_{s \in 2^{\mathcal{A}^\epsilon}, a \in s} \neg P_s^{r(O)}(x_{i(O)}) & a \in \mathcal{A}^\epsilon \\ \neg x_{i(O)} \in A^{r(O)} & \text{otherwise.} \end{cases} \\
f(\alpha * \beta, O) =& f(\alpha,O) \wedge f(\beta,O) \\
f(\alpha + \beta, O) =& f(\alpha,O) \vee f(\beta,O) \\
f(\Diamond\alpha, O) =& \exists x_{i(O)+1}. f(\alpha, (O, i(O)+1)) \wedge x_{i(O)+1} \in T^{r(O)} \\
f(\Box\alpha, O) =& \forall x_{i(O)+1}. \left(f(\alpha, (O, i(O)+1)) \wedge x_{i(O)+1} \in T^{r(O)}\right) \vee x_{i(O)+1} \notin T^{r(O)} \\
f(\boxplus^\varpi \alpha, O) =& \exists P^{(r(O),\varpi)} : T(\phi_\varpi(P^{r(O)}, P^{(r(O),\varpi)}, x_{i(O)})) \wedge \\
& \exists T^{(r(O),\varpi)} : T(\psi_\varpi(T^{r(O)}, T^{(r(O),\varpi)}, x_{i(O)})) \wedge f(\alpha, (O,\varpi))
\end{aligned}$$

where $i(O)$ is the maximum integer in O and $r(O)$ is the sequence O reduced to the entries that are in W. We set $T^\epsilon = T$ and $P^\epsilon = P(v)$.

For a weighted LARS formula α we define its translation to weighted MSO by

$$f(\alpha) = f(\alpha, \epsilon).$$

Faithfulness is proven as invariance under translation:

Lemma A.5.4 (Invariance of the formal power series under translation)**.** *Let α be some weighted LARS formula over semiring $\mathcal{R}$. Then*

$$\forall S,t : [\![\alpha]\!]_\mathcal{R}(S,t) = [\![f(\alpha)]\!]_\mathcal{R}(\sigma(S,t)),$$

where $\sigma(S,t) = (\sigma_1(S,t), \sigma_2(S,t))$ is defined as above.

Proof. We prove our lemma by structural induction on α.
Case $\alpha \in \{a, \neg a, k\}$: $\surd$.
Case $\alpha \in \{\alpha_1 * \alpha_2, \alpha_1 + \alpha_2\}$: $\surd$ by inductive hypothesis.
Due to the similarities between weighted MSO and weighted LARS a translation for most logical operators in LARS is straight forward. Difficulties only occur, when we have quantification with $\Box, \Diamond$, since we then only want to quantify over the times/positions in the word, that are within the substream given by the window operator. This is achieved by putting restrictions on the quantified variables by using the MSO variables $T^{r(O)}$.
In the case of universal quantification we need to make sure that we obtain $e_{\otimes}$, when we quantify over an empty set of time points. □

Additionally, we require a translation from unweighted LARS to unweighted MSO.

Lemma A.5.5 (LARS programs and regular languages)**.** *Let* Π *some LARS program using only MSO-definable window functions, then there exists an MSO formula* Φ *such that for all data streams* D *for* Φ_Π *it holds that if* S *is an interpretation stream of* D*, then*

$$\forall t \in T : S \in \mathcal{AS}(\Pi, D, t) \iff \sigma(S, t) \models_{MSO} \Phi_\Pi$$

where σ *is defined as in Lemma A.5.4 and* $\models_{MSO}$ *is satisfaction w.r.t. MSO semantics.*

Proof. Given an unweighted rule $r = \alpha \leftarrow \beta_1, \ldots, \beta_n, \neg\beta_{n+1}, \ldots, \neg\beta_{n+m}$, we define its second order translation by

$$f(r) = f(\alpha) \vee \neg f(\beta_1) \vee \cdots \vee \neg f(\beta_{n+m}).$$

This translation is extended to programs Π via

$$f(\Pi) = \bigwedge_{r \in \Pi} f(r).$$

Now let $f'(\Pi)$ the same translation except that every monadic second order variable A is replaced with A' whenever it occurs in the body of a rule. Then

$$\Phi_\Pi(\mathbf{A}) := f(\Pi) \wedge \forall A'_1 \ldots \forall A'_k . \neg f'(\Pi) \vee \neg (A'_1, \ldots, A'_k) < (A_1, \ldots, A_k)$$

is an unweighted MSO formula characterizing if an assignment to the MSO variables A_i corresponds to an answer stream of Π, similarly to what was done in [EG97]. $(A'_1, \ldots, A'_k) < (A_1, \ldots, A_k)$ defined as in [EG97] is true iff $\mathbf{A}$ corresponds to a stream that is a strict substream of the stream corresponding to $\mathbf{A}'$.

Therefore if S is an interpretation stream of D it holds that

$$S \in \mathcal{AS}(\Pi, D, t) \iff \sigma(S, t) \models_{\text{MSO}} \Phi_\Pi$$

for an arbitrary time point t. □

Since we want to have a weighted MSO formula in the end, we need to translate the obtained unweighted MSO formula into a weighted one. For this, we can use previous work of Droste and Gastin [DG07].

Definition A.5.6 (Recognizable Power Series, Recognizable Step Function [DG07])**.** *A formal power series $\phi : A^* \to R$ is* recognizable *if there exists a weighted automaton $\mathcal{A}$, s.t.*

$$\forall w \in A^* : \|\mathcal{A}\|(w) = \phi(w).$$

Furthermore, ϕ is a recognizable step function *if*

$$\phi = \bigoplus_{i=1}^{n} k_i \mathbb{1}_{\mathcal{L}_i},$$

for recognizable languages $\mathcal{L}_i \subseteq A^*$.

Lemma A.5.7 (Classical MSO in weighted MSO [DG07])**.** *For each unweighted MSO formula ϕ there exists a weighted MSO formula $T(\phi)$, s.t.*

$$[\![T(\phi)]\!] = \mathbb{1}_{\mathcal{L}(\phi)},$$

where $\mathcal{L}(\phi)$ is the regular language defined by ϕ and $\mathbb{1}_{\mathcal{L}(\phi)}$ is equal to $e_\otimes$ for all words $w \in \mathcal{L}(\phi)$ and equal to $e_\oplus$ otherwise.

With these ingredients given, we can finally proceed with the original proof.

Proof of Theorem 65. Let Φ_Π be the MSO formula constructed in Lemma A.5.5. If we interpret Φ_Π as a weighted formula it is not restricted and therefore not necessarily recognizable, nor does it necessarily have semantics corresponding to the boolean semantics. However, if we replace Φ_Π by $T(\Phi_\Pi)$ both of the above conditions hold. We know that $T(\Phi_\Pi)$ captures the boolean semantics, according to Lemma A.5.7 and we further know that since $\mathcal{L}(\Phi_\Pi)$ is recognizable language (in the unweighted context) it is also true, that $\mathbb{1}_{\mathcal{L}(\Phi_\Pi)}$, which is the semantics of $T(\Phi_\Pi)$, is recognizable [DG07](Lemma 2.1 b)).

Then since α is recognizable, $f(\alpha)$ is also recognizable. It follows, according to [DG07] (Lemma 4.2), that $T(\Phi)(\mathbf{A}) \wedge f(\alpha)$ is also recognizable and therefore also $\exists \mathbf{A}.T(\Phi)(\mathbf{A}) \wedge f(\alpha)$, by [DG07] (Lemma 4.3).

Therefore, there exists a weighted automaton $\mathcal{A}$ that recognizes it. What is left to show is that

$$\forall D, t : \mu(\Pi, D, t) = \|\mathcal{A}\|(\sigma(D, t)),$$

holds for this automaton. We use the fact proven in Lemma A.5.4, that

$$\forall S, t : [\![\alpha]\!]_{\mathcal{R}}(S, t) = [\![f(\alpha)]\!]_{\mathcal{R}}(\sigma(S, t)).$$

Since σ is a bijection between the combination of a streams and time points, and MSO assignments we have that for all D, t

$$\begin{aligned}\mu(\Pi, D, t) &= \bigoplus_{S \in \mathcal{AS}(\Pi, D, t)} [\![\alpha]\!]_{\mathcal{R}}(S, t) \\ &= \bigoplus_{S \in \mathcal{AS}(\Pi, D, t)} [\![f(\alpha)]\!]_{\mathcal{R}}(\sigma(S, t)) \\ &= \bigoplus_{S \text{ interpretation stream of } D} [\![T(\Phi_\Pi)(\mathbf{A}) \wedge f(\alpha)]\!]_{\mathcal{R}}(\sigma(S, t))\end{aligned}$$

Instead of summing over the streams that are interpretation streams we can equivalently sum over all possible assignments to the second order variables $\mathbf{A}$. This however is equivalent to quantifying $\mathbf{A}$ existentially:

$$\begin{aligned}\mu(\Pi, D, t) =& [\![\exists \mathbf{A}. T(\Phi_\Pi)(\mathbf{A}) \wedge f(\alpha)]\!]_{\mathcal{R}}(\sigma(D, t)) \\ =& \|\mathcal{A}\|(\sigma(D, t))\end{aligned}$$

□

A.6 Computational Complexity of LARS Measures Over Propositional Variables ↪ Lemma 67 and Theorem 68

Lemma 67 (FPSpace(poly)-hardness)**.** *Let $\mathcal{Q} = (2^{\mathbb{N}}, \min, \cup, \mathbb{N}, \emptyset)$, where the minimum is taken w.r.t. to the order $\succ$, where $X \succ Y$ holds iff*

$$\exists n \in X : (\forall n' < n : n' \in X \iff n' \in Y) \wedge n \in X \setminus Y.$$

That is, $X \succ Y$ iff the smallest number that X and Y disagree on is in X.

*The problem **EVAL-wLARS** over $\mathcal{Q}$ is FPSpace(poly)-hard.*

Proof. We reduce the problem QBF-sat-search of finding a satisfying assignment for the free variables of a QBF formula ϕ, given that it is satisfiable to **EVAL-wLARS** over semiring $\mathcal{Q}$. Since it is know that QBF-sat-search is FPSpace(poly)-complete [HSH12], this suffices.
Let

$$\Phi = Q_1 x_1 \dots Q_n x_n \phi(x_1, \dots, x_n, y_1, \dots, y_m)$$

some QBF formula with free variables $y_1, \dots, y_m$. We construct the input for **EVAL-wLARS** over semiring $\mathcal{Q}$ as follows:

$$\begin{aligned}\alpha =& \Diamond \boxplus^{set:y_1} \dots \Diamond \boxplus^{set:y_m} \\ & \bigwedge_{i=1}^{m} (\neg y_i \vee (y_i \wedge \{i\})) \wedge \alpha'(y_1, \dots, y_m)\end{aligned}$$

$$\begin{aligned}\alpha' =& W_1 \boxplus^{set:x_1} \dots W_n \boxplus^{set:x_n} \phi(x_1, \dots, x_n, y_1, \dots, y_m)\\ S =& (\{0,1\}, x \mapsto \{x_1, \dots, x_n, y_1, \dots, y_m\})\\ t =& 0\end{aligned}$$

Somewhat similarly to the proof of PSPACE hardness of model checking in [BDE18]. The window function $set : z$ for some propositional variable z is defined by

$$set : z((T, v), t) = (T, v'),$$
$$v'(j) = \begin{cases} v(j) \setminus \{z\} & \text{if } t = 0,\\ v(j) & \text{if } t \neq 0.\end{cases}$$

Further

$$W_i = \begin{cases} \Diamond & \text{if } Q_i = \exists,\\ \Box & \text{if } Q_i = \forall.\end{cases}$$

We can argue analogously to Beck, Dao-Tran, and Eiter [BDE18], that using $W\boxplus^{set:z}$ we can model quantification over the possible boolean values of the variable z. Therefore if we evaluate the first part of α we get

$$\begin{aligned}&[\![\alpha]\!]_{\mathcal{R}}(S, t)\\ =& \min_{\overline{y}\in\{0,1\}^m} \left[\!\!\left[\bigwedge_{i=1}^{m} (\neg y_i \vee (y_i \wedge \{i\})) \wedge \alpha'(y_1, \dots, y_m) \right]\!\!\right]_{\mathcal{R}} (S, 0)\\ =& \min_{\overline{y}\in\{0,1\}^m} \{i \in \{1, \dots, m\} \mid y_i = 1\}\\ &\cup [\![\alpha'(y_1, \dots, y_m)]\!]_{\mathcal{R}} (S, 0).\end{aligned}$$

If α' is satisfied by the given assignment $\overline{y}$, we can see that α' evaluates to $\emptyset$, otherwise it evaluates to $\mathbb{N}$. Therefore we obtain

$$\begin{aligned}&[\![\alpha]\!]_{\mathcal{R}}(S, t)\\ =& \min_{\overline{y}\in\{0,1\}^m} \begin{cases} \{i \in \{1, \dots, m\} \mid y_i = 1\} & \text{if } \models \alpha'(\overline{y}),\\ \mathbb{N} & \text{otherwise.}\end{cases}\end{aligned}$$

Therefore we obtain $\mathbb{N}$ as the result iff none of the assignments satisfy α'. Otherwise we get a set $I \subseteq \{1, \dots, m\}$, s.t. for the vector $\overline{y}(I)$ given by

$$\overline{y}(I)_i = \begin{cases} 1 & \text{if } i \in I,\\ 0 & \text{if } i \notin I.\end{cases}$$

it holds that $\alpha'(\overline{y}(I))$ is satisfied and there is no set $I' \subseteq \{1, \dots, m\}$ s.t. $I \succ I'$ and $\alpha'(\overline{y}(I'))$ is satisfied. That means among other things, that $\overline{y}(I)$ is a satisfying assignment for Φ. We can therefore reduce QBF-sat-search to **EVAL-wLARS** over semiring $\mathcal{Q}$. □

Theorem 68 (Complexity of Evaluation I)**.** *Let $\mathcal{R}$ be a fixed semiring that is efficiently encoded by e, and let $k \in \mathbb{N}$ be a fixed constant. Then for any weighted LARS formula α over $\mathcal{R}$ s.t. $qdepth(\alpha) \leq k$, we can calculate $[\![\alpha]\!]_{\mathcal{R}}(S,t)$ in polynomial time in the size of α and S.*

Proof. Proof by structural induction on the formula α, with induction invariant that $t(\alpha)$ the time needed is in $\mathcal{O}(N^{n\cdot(k+1)})$, where N is the size of the input, $n \in \mathbb{N}$ is a constant not depending on the input and k is the quantifier nesting depth. Further, $s(\alpha)$ the size of the representation of the obtained value, i.e. $\|[\![\alpha]\!]_{\mathcal{R}}(S,t)\|_e$, is in $\mathcal{O}(N \cdot N^k)$.‘

Base Cases:

- $\alpha = e(k)$: Then one can evaluate the expression by simply returning $e(k)$. This is feasible in polynomial time. The size of the output is linear in the size of the input.
- $\alpha = p$: We simply check if $p \in v(t)$ and return $e_\oplus$ or $e_\otimes$ accordingly. This is possible in constant time and the size of the output is also bounded by a constant.

We have shown the invariant for all formulae up to a certain structural complexity.

Induction Step:

- $\alpha = \neg\beta$: We know that for β the invariant holds, therefore we can check in time bounded polynomially in the size of the formula and exponentially in the quantifier nesting depth, whether $[\![\beta]\!]_{\mathcal{R}}(S,t) = e_\oplus$ and output $e_\oplus$ or $e_\otimes$ accordingly. The size of the output is again bounded by a constant.
- $\alpha = \beta_1 + \beta_2$: We know that the invariant holds for β_1, β_2. Further $t(\alpha) = t(\beta_1) + t(\beta_2) + x$, where x is the time needed for addition of the results for β_1 and β_2. We know that x is polynomial in $s(\beta_1) + s(\beta_2)$, which we know to be in $\mathcal{O}(N^{k+1})$. Therefore $x \in \mathcal{O}(N^{(k+1)\cdot l}$, where l is the degree of the polynomial bounding the time needed to add two numbers. It follows that $t(\alpha) \in \mathcal{O}(N^{n\cdot k})$. For $s(\alpha)$ we can see that

$$\begin{aligned} s(\alpha) &= \|[\![\alpha]\!]_{\mathcal{R}}(S,t)\|_e \\ &\leq p(1)\max(\|[\![\beta_1]\!]_{\mathcal{R}}(S,t)\|_e, \|[\![\beta_2]\!]_{\mathcal{R}}(S,t)\|_e) \end{aligned}$$

  And therefore $s(\alpha) \in \mathcal{O}(N^{k+1})$.
- $\alpha = \beta_1 * \beta_2$: The proof works analogously to the proof for the case $\alpha = \beta_1 + \beta_2$.
- $\alpha = -\beta$: Follows immediately from the induction hypothesis for β and efficient encodedness.
- $\alpha = \beta^{-1}$: Follows immediately from the induction hypothesis for β and efficient encodedness.

- $\alpha = @_{t'}\beta$: Follows immediately from the induction hypothesis for β evaluated at t'.
- $\alpha = \Diamond\beta$: The quantifier nesting depth of α is one higher than that of β. Therefore, we need to show that if $t(\beta) \in \mathcal{O}(N^{n\cdot(k+1)})$ and $s(\beta) \in \mathcal{O}(N^{k+1})$, then $t(\alpha) \in \mathcal{O}(N^{n\cdot(k+2)})$ and $s(\alpha) \in \mathcal{O}(N^{k+2})$. We start with the second statement.

  $$s(\alpha) \leq p(N) \max_{t\in T} \|[\![\beta]\!]_{\mathcal{R}}(S,t)\|_e$$

  Therefore, $s(\alpha) \in \mathcal{O}(N^{k+2})$. The first statement can be proven similarly to the case for the connective $+$: We have to evaluate β for all $t \in T$ and sum up the results. The time needed to perform all evaluations is in $\mathcal{O}(N^{n\cdot k+1})$. The time needed to sum up all the values is polynomial in the size of the values, which do increase in size, but for every sum it holds that the values are in $\mathcal{O}(N^{k+2})$, so $t(\alpha) \in \mathcal{O}(N^{(k+2)\cdot l+1})$ and therefore $t(\alpha) \in \mathcal{O}(N^{(k+2)\cdot(l+1)})$.
- $\alpha = \Box\beta$: The proof works analogously to the proof for the case $\alpha = \Diamond\beta$.
- $\alpha = \boxplus^{\varpi}\beta$: The evaluation of ϖ is feasible in polynomial time. The claim follows immediately from the induction hypothesis for β evaluated using $\varpi(S,t)$ instead of S. The derived bounds are sufficient, since $\varpi(S,t)$ needs to be a substream of S.
- $\alpha = \triangleright\beta$: The claim follows immediately from the induction hypothesis for β evaluated using S^* instead of S.

If k is now a constant, then we derived that the evaluation can be performed in polynomial time in the size of the formula and the interpretation. □

APPENDIX B

Full Proofs: Complexity of Counting over Semirings

B.1 Prefix Normal Form ↪ Lemma 81

Lemma 81 (Prefix Normal Form)**.** *For every ΣBF α over a semiring $\mathcal{R}$ there exists a ΣBF β over $\mathcal{R}$ such that*

(i) $\beta = \Sigma v_1 \dots \Sigma v_n \gamma$, *where γ is quantifier free,*

(ii) β *can be constructed from α in polynomial time, and*

(iii) $[\![\alpha]\!]_{\mathcal{R}}(\emptyset) = [\![\beta]\!]_{\mathcal{R}}(\emptyset)$, *i.e., α and β evaluate to the same value.*

Proof. Let α be a ΣBF over semiring $\mathcal{R}$. We show that we can always push out the sum quantifiers.
Let $\alpha = (\Sigma v \alpha_1) * \alpha_2$. W.l.o.g. we can assume that α_2 does not contain v. Then

$$\begin{aligned}[\![\alpha]\!]_{\mathcal{R}}(\emptyset) &= [\![(\Sigma v \alpha_1) * \alpha_2]\!]_{\mathcal{R}}(\emptyset) \\ &= [\![\Sigma v \alpha_1]\!]_{\mathcal{R}}(\emptyset) \otimes [\![\alpha_2]\!]_{\mathcal{R}}(\emptyset) \\ &= ([\![\alpha_1]\!]_{\mathcal{R}}(\{v\}) \oplus [\![\alpha]\!]_{\mathcal{R}}(\{\neg v\})) \otimes [\![\alpha_2]\!]_{\mathcal{R}}(\emptyset)\end{aligned}$$

Since addition distributes over multiplication and α_2 does not contain v, we get

$$\begin{aligned}[\![\alpha]\!]_{\mathcal{R}}(\emptyset) &= [\![\alpha_1]\!]_{\mathcal{R}}(\{v\}) \otimes [\![\alpha_2]\!]_{\mathcal{R}}(\emptyset) \oplus [\![\alpha_1]\!]_{\mathcal{R}}(\{\neg v\}) \otimes [\![\alpha_2]\!]_{\mathcal{R}}(\emptyset) \\ &= [\![\alpha_1]\!]_{\mathcal{R}}(\{v\}) \otimes [\![\alpha_2]\!]_{\mathcal{R}}(\{v\}) \oplus [\![\alpha_1]\!]_{\mathcal{R}}(\{\neg v\}) \otimes [\![\alpha_2]\!]_{\mathcal{R}}(\{\neg v\}) \\ &= [\![\alpha_1 * \alpha_2]\!]_{\mathcal{R}}(\{v\}) \oplus [\![\alpha_1 * \alpha_2]\!]_{\mathcal{R}}(\{\neg v\})\end{aligned}$$

$$= [\![\Sigma v(\alpha_1 * \alpha_2)]\!]_{\mathcal{R}}(\emptyset)$$

So we can choose $\beta = \Sigma v(\alpha_1 * \alpha_2)$.
Let $\alpha = \alpha_1 * (\Sigma v \alpha_2)$. This case works analogously to the previous one.
Let $\alpha = (\Sigma v \alpha_1) + \alpha_2$. Then $\beta = \Sigma v(\alpha_1 + \alpha_2 * v)$ has the following semantics w.r.t. $\emptyset$

$$\begin{aligned}
[\![\beta]\!]_{\mathcal{R}}(\emptyset) &= [\![\Sigma v(\alpha_1 + \alpha_2 * v)]\!]_{\mathcal{R}}(\emptyset) \\
&= [\![\alpha_1 + \alpha_2 * v]\!]_{\mathcal{R}}(\{v\}) \oplus [\![\alpha_1 + \alpha_2 * v]\!]_{\mathcal{R}}(\{\neg v\}) \\
&= [\![\alpha_1 + \alpha_2]\!]_{\mathcal{R}}(\{v\}) \oplus [\![\alpha_1]\!]_{\mathcal{R}}(\{\neg v\}) \\
&= [\![\alpha_1]\!]_{\mathcal{R}}(\{v\}) \oplus [\![\alpha_1]\!]_{\mathcal{R}}(\{\neg v\}) \oplus [\![\alpha_2]\!]_{\mathcal{R}}(\{v\})
\end{aligned}$$

Since w.l.o.g. v does not occur in α_2 we have

$$\begin{aligned}
[\![\beta]\!]_{\mathcal{R}}(\emptyset) &= [\![\alpha_1]\!]_{\mathcal{R}}(\{v\}) \oplus [\![\alpha_1]\!]_{\mathcal{R}}(\{\neg v\}) \oplus [\![\alpha_2]\!]_{\mathcal{R}}(\{v\}) \\
&= [\![\Sigma v \alpha_1]\!]_{\mathcal{R}}(\emptyset) \oplus [\![\alpha_2]\!]_{\mathcal{R}}(\emptyset) \\
&= [\![(\Sigma v \alpha_1) + \alpha_2]\!]_{\mathcal{R}}(\emptyset)
\end{aligned}$$

So we can choose $\beta = \Sigma v(\alpha_1 + \alpha_2 * v)$.
The case $\alpha = \alpha_1 + \Sigma x \alpha_2$ works analogously.
This shows that we can always push the Σ quantifier to the top in polynomially many steps in the number of occurrences of $*, +$ in α. Furthermore, as we only add $*v$ to the formula at most once for every combination of Σ and $+$ in α, the size of β is also polynomial in α. □

B.2 NP($\mathcal{R}$)-completeness and Karp reducibility

We prove NP($\mathcal{R}$)-completeness of SAT($\mathcal{R}$) and show that SAT($\mathcal{R}$), SumProd($\mathcal{R}$), AMC, SCSP and ΣFO-Eval($\mathcal{R}$) are Karp-reducible to one another.

B.2.1 NP($\mathcal{R}$)-completeness of SAT($\mathcal{R}$) ↪ Theorem 88

Theorem 88. *SAT($\mathcal{R}$) is NP($\mathcal{R}$)-complete w.r.t. Karp s-reductions for every commutative semiring $\mathcal{R}$.*

Proof. Containment follows from Algorithm 1 and Proposition B.2.1.

For hardness we generalize the Cook-Levin Theorem. So let $M = (\mathcal{R}, R', Q, \Sigma, \iota, \sqcup, \delta)$ be a polynomial time SRTM and $x \in (\Sigma \cup R)^*$ be the input for which we want to compute the output of M.

We define the following propositional variables:

- $T_{i,j,k}$, which is true if tape cell i contains symbol j at step k of the computation.
- $H_{i,k}$, which is true if the M's read/write head is at tape cell i at step k of the computation.
- $Q_{q,k}$, which is true if M is in state q at step k of the computation.

Furthermore, we need the following semiring values

- r_i, which is the i^{th} semiring value of the input x, and
- $\{r_{s+1}, \ldots, r_m\} = R'$, which are the constant semiring values that M has access to. Here, s is the number of semiring values in x.

Since M is a polynomial time SRTM, we can assume the existence of a polynomial p such that $p(n)$ bounds the number of transitions of M on any input of length n.

Given a finite set I and a family $(\beta_i)_{i\in I}$ of weighted formulas, we use the following shorthand:

$$\Sigma_{i\in I}\beta_i = \begin{cases} e_\oplus & \text{if } I = \emptyset \\ \beta_{i^*} + \Sigma_{i\in I\setminus\{i^*\}}\beta_i & \text{if } i^* \in I \end{cases}.$$

Note that this is well defined, since I is finite and addition is commutative and associative.

We define a weighted QBF $\Sigma\mathbf{T}\ \Sigma\mathbf{H}\ \Sigma\mathbf{Q}\ \alpha$, where $\Sigma\mathbf{T}$, $\Sigma\mathbf{H}$, and $\Sigma\mathbf{Q}$ correspond to the (sum) quantification of all variables $T_{i,j,k}$, $H_{i,k}$, and $Q_{q,k}$, respectively, and α is the product (i.e. connected with $*$) of the following subformulas

1. $T_{i,j,0}$
   Variable ranges: $0 \le i < n$
   For each tape cell i that initially contains symbol $j \in \Sigma$ or $j = r$ when it contains the semiring value r.
2. $T_{i,\sqcup,0}$
   Variable ranges: $-p(n) \le i < 0$ or $n \le i \le p(n)$
   Each tape cell i outside of the ones that contain the input contains $\sqcup$.
3. $Q_{\iota,0}$
   The initial state of M is ι.
4. $H_{0,0}$
   The initial position of the head is 0.
5. $\neg T_{i,j,k} + T_{i,j,k} * \neg T_{i,j',k}$
   Variable ranges: $-p(n) \le i \le p(n), j \in \Sigma \cup \{r_1, \ldots, r_s\}, 0 \le k \le p(n)$
   There is at most one symbol per tape cell.

6. $\Sigma_{j\in\Sigma\cup\{r_{i_1},\ldots,r_{i_m}\}} T_{i,j,k}$
   Variable ranges: $-p(n) \leq i \leq p(n), 0 \leq k \leq p(n)$
   There is at least one symbol per tape cell.

7. $\neg T_{i,j,k} + T_{i,j,k} * \neg T_{i,j',k+1} + T_{i,j,k} * T_{i,j',k+1} * H_{i,k}$
   Variable ranges: $-p(n) \leq i \leq p(n), j \neq j' \in \Sigma \cup \{r_1, \ldots, r_s\}, 0 \leq k < p(n)$
   Tape remains unchanged unless written.

8. $\neg Q_{q,k} + Q_{q,k} * \neg Q_{q',k}$
   Variable ranges: $q \neq q' \in Q, 0 \leq k \leq p(n)$
   There is at most one state at a time.

9. $\neg H_{i,k} + H_{i,k} * \neg H_{i',k}$
   Variable ranges: $i \neq i', -p(n) \leq i \leq p(n), -p(n) \leq i' \leq p(n), 0 \leq k \leq p(n)$
   There is at most one head position at a time.

10. $\neg H_{i,k} + H_{i,k} * \neg Q_{q,k} + H_{i,k} * Q_{q,k} * \neg T_{i,\sigma,k} +$
    $H_{i,k} * Q_{q,k} * T_{i,\sigma,k} * \Sigma_{((q,\sigma),(q',\sigma'),d,r)\in\delta'} H_{i+d,k+1} * Q_{q',k+1} * T_{i,\sigma',k+1} * r$
    Variable ranges: $-p(n) \leq i \leq p(n), 0 \leq k < p(n)$ and $q \in Q$, $\sigma \in \Sigma \cup \{r_{i_1}, \ldots, r_{i_m}\}$ s.t. there exist q', σ', d, r s.t. $((q,\sigma),(q',\sigma'),d,r) \in \delta$.
    Possible transitions at computation step k when head is at position i with their respective weight.
    Here, we use

    $$\delta' = \{((q,\sigma),(q',\sigma'),d,r) \in \delta \mid \sigma, \sigma' \in \Sigma \cup \{r_1, \ldots, r_m\}\}$$

    since we only need to take into account the transitions for letters from the alphabet Σ, the semiring values $\{r_1, \ldots, r_s\}$ in the input and the constants $\{r_{s+1}, \ldots, r_m\}$. With this restriction $|\delta'|$ is finite and therefore the size of the sum is bounded by a constant that only depends on the SRTM.

11. $\neg H_{i,k} + H_{i,k} * \neg Q_{q,k} + H_{i,k} * Q_{q,k} * \neg T_{i,\sigma,k}$
    $+H_{i,k} * Q_{q,k} * T_{i,\sigma,k} * H_{i,k+1} * Q_{q,k+1} * T_{i,\sigma,k+1}$
    Variable ranges: $-p(n) \leq i \leq p(n), 0 \leq k < p(n)$ and $q \in Q$, $\sigma \in \Sigma \cup \{r_1, \ldots, r_m\}$ s.t. there do not exist q', σ', d, r s.t. $((q,\sigma),(q',\sigma'),d,r) \in \delta$.
    The machine computation has ended. This is included so that when the machine has reached a final state it stays the same until k reaches $p(n)$.

In order to prove correctness, i.e., that the value of M on x is equal to $[\![\Sigma\mathbf{T}\ \Sigma\mathbf{H}\ \Sigma\mathbf{Q}\ \alpha]\!]_{\mathcal{R}}(\emptyset)$, we prove two claims.

(i) For each interpretation $\mathcal{I}$ of the variables $T_{i,j,k}, H_{i,k}, Q_{q,k}$ for $-p(n) \leq i \leq p(n), 0 \leq k < p(n)$ and $q \in Q$ such that $\mathcal{I}$ does not correspond to a computation path of M on x, it holds that $[\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\oplus}$.

(ii) For each interpretation $\mathcal{I}$ of the variables $T_{i,j,k}, H_{i,k}, Q_{q,k}$ for $-p(n) \leq i \leq p(n), 0 \leq k < p(n)$ and $q \in Q$ such that $\mathcal{I}$ corresponds to a computation path π of M on x along configurations $c_1^\pi \overset{r^{(\pi_1)}}{\to} \dots \overset{r^{(\pi_{n(\pi)-1})}}{\to} c_{n(\pi)}^\pi$ it holds that

$$[\![\alpha]\!]_\mathcal{R}(\mathcal{I}) = \bigoplus_{\pi', \text{ s.t.} c_i^\pi = c_i^{\pi'}} r^{(\pi'_1)} \otimes \dots \otimes r^{(\pi'_{n(\pi')-1})}.$$

If both claims hold, then it follows that $[\![\Sigma \mathbf{T}\ \Sigma \mathbf{H}\ \Sigma \mathbf{Q}\ \alpha]\!]_\mathcal{R}(\emptyset)$ is equal to the sum of $[\![\alpha]\!]_\mathcal{R}(\mathcal{I})$ over all interpretations $\mathcal{I}$ such that $\mathcal{I}$ corresponds to a computation path. For each of them, we know that the weight of the path is the product of the weights of the taken transitions, according to (ii). Since the value of M on x is equal to the sum of the weights of the paths, this implies correctness of the reduction.

We proceed to prove the claims. For this, we first generally show that the added subformulas 1. to 11. enforce their given purpose.

For 1. to 4. this is clear: In order for $[\![\alpha]\!]_\mathcal{R}(\mathcal{I})$ to be unequal to $e_\oplus$, the variables need to be included in the interpretation.

5. and 6. together ensure that at each time step there is exactly one symbol in each tape cell. So assume that $T_{i,j,k}$ and $T_{i,j',k}$ are in $\mathcal{I}$. Then

$$[\![\neg T_{i,j,k} + T_{i,j,k} * \neg T_{i,j',k}]\!]_\mathcal{R}(\mathcal{I}) = e_\oplus \oplus e_\oplus = e_\oplus,$$

and so $[\![\alpha]\!]_\mathcal{R}(\mathcal{I}) = e_\oplus$. Assume alternatively that there are i, k such that for no j the variable $T_{i,j,k}$ is in $\mathcal{I}$. Then

$$[\![\Sigma_{j \in \Sigma \cup \{r_1, \dots, r_s\}} T_{i,j,k}]\!]_\mathcal{R}(\mathcal{I}) = e_\oplus.$$

7. ensures that the tape remains unchanged unless written, i.e., unless the head is at position i at step k the value of the tape cell i stays the same. So assume that $H_{i,k}$ is not in $\mathcal{I}$ but $T_{i,j,k}$ and $T_{i,j',k}$ for $j \neq j'$ are in $\mathcal{I}$. Then

$$[\![\neg T_{i,j,k} + T_{i,j,k} * \neg T_{i,j',k+1} + T_{i,j,k} * T_{i,j',k+1} * H_{i,k}]\!]_\mathcal{R}(\mathcal{I}) = e_\oplus \oplus (e_\otimes \otimes e_\oplus) \oplus (e_\otimes \otimes e_\otimes \otimes e_\oplus) = e_\oplus.$$

8. ensures that there is at most one state at a time by analogous reasoning to 5.

9. ensures that there is at most one head position at a time by analogous reasoning to 5.

10. models the possible transitions at computation step k when the head is at position i including their respective weights, *if there is a possible transition* for the given state and tape cell entry. Otherwise, this subformula is not added but the one in 11. is. So assume that $H_{i,k}, Q_{q,k}, T_{i,\sigma,k} \in \mathcal{I}$. Then the value of the subformula is

$$[\![\neg H_{i,k} + H_{i,k} * \neg Q_{q,k} + H_{i,k} * Q_{q,k} * \neg T_{i,\sigma,k}]\!]_\mathcal{R}(\mathcal{I}) \oplus$$
$$[\![H_{i,k} * Q_{q,k} * T_{i,\sigma,k} * \Sigma_{((q,\sigma),(q',\sigma'),d,r) \in \delta'} H_{i+d,k+1} * Q_{q',k+1} * T_{i,\sigma',k+1} * r]\!]_\mathcal{R}(\mathcal{I})$$

$$= e_\oplus \oplus e_\otimes \otimes [\![\Sigma_{((q,\sigma),(q',\sigma'),d,r)\in\delta'} H_{i+d,k+1} * Q_{q',k+1} * T_{i,\sigma',k+1} * r]\!]_{\mathcal{R}}(\mathcal{I})$$
$$= [\![\Sigma_{((q,\sigma),(q',\sigma'),d,r)\in\delta'} H_{i+d,k+1} * Q_{q',k+1} * T_{i,\sigma',k+1} * r]\!]_{\mathcal{R}}(\mathcal{I})$$

This means that the expression evaluates to the sum of all weights of the transitions we take. Note that in order for two transitions $((q,\sigma),(q'_1,\sigma'_1),d_1,r_1)$, $((q,\sigma),(q'_2,\sigma'_2),d_2,r_2)$ to be different at least one of $q'_1 \neq q'_2$, $\sigma'_1 \neq \sigma'_2$, $d_1 \neq d_2$ or $r_1 \neq r_2$ needs to hold. If $q'_1 \neq q'_2$, $\sigma'_1 \neq \sigma'_2$ or $d_1 \neq d_2$ then one of 5., 8., or 9. is falsified. Thus, in this case, we can take multiple transitions if they differ in the weights only. On the other hand, we must take at least one transition, since otherwise the whole sum evaluates to $e_\oplus$. It follows that we transition to exactly one new configuration to obtain a non-zero value for α. In that case, we have $H_{i+d,k+1}, Q_{q',k+1}, T_{i,\sigma',k+1} \in \mathcal{I}$ for the corresponding transition(s) $((q,\sigma),(q',\sigma'),d,r) \in \delta'$ and

$$[\![\Sigma_{((q,\sigma),(q',\sigma'),d,r)\in\delta'} H_{i+d,k+1} * Q_{q',k+1} * T_{i,\sigma',k+1} * r]\!]_{\mathcal{R}}(\mathcal{I}) = \bigoplus_{((q,\sigma),(q',\sigma'),d,r)\in\delta'} r.$$

11. models that if at computation step k when the head is at position i *there is no possible transition* for the given state q and tape cell entry σ, then the head position, state and tape cell entries stay the same. Otherwise, this subformula is not added but one in 10. is. So assume that $H_{i,k}, Q_{q,k}, T_{i,\sigma,k} \in \mathcal{I}$. Then the value of the subformula is

$$[\![\neg H_{i,k} + H_{i,k} * \neg Q_{q,k} + H_{i,k} * Q_{q,k} * \neg T_{i,\sigma,k}]\!]_{\mathcal{R}}(\mathcal{I}) \oplus$$
$$[\![H_{i,k} * Q_{q,k} * T_{i,\sigma,k} * H_{i,k+1} * Q_{q,k+1} * T_{i,\sigma,k+1}]\!]_{\mathcal{R}}(\mathcal{I})$$
$$= e_\oplus \oplus e_\otimes \otimes [\![H_{i,k+1} * Q_{q,k+1} * T_{i,\sigma,k+1}]\!]_{\mathcal{R}}(\mathcal{I})$$

This means that the expression evaluates to $e_\otimes$, if $H_{i,k+1}, Q_{q,k+1}, T_{i,\sigma,k+1} \in \mathcal{I}$ and evaluates to $e_\oplus$, otherwise. Thus, the formula enforces the desired constraint: if we are in a configuration without further transitions we stay in it until the time limit $p(n)$ is reached. Notably, this does not influence the weight, since we always obtain a factor of $e_\otimes$.

Putting things together, we see that 8. and 10./11. together ensure that there is exactly one state at each time. Similarly, 9. and 10./11. together ensure that there is exactly one head position at each time. This together with the constraints associated originally with 1. to 11. show that the desired claims and therefore also the theorem hold. □

Proposition B.2.1. *The sum, using $\oplus$, of the results of all execution paths for a call to $\textsc{Eval}_{\mathcal{R}}(\alpha,\mathcal{I})$ is equal to $[\![\alpha]\!]_{\mathcal{R}}(\mathcal{I})$.*

Proof. We proceed by structural induction on α.

- Case $\alpha = k$:
  The algorithm returns k, therefore the statement is true.
- Case $\alpha = l, l \in \{v, \neg v\}$:
  The algorithm returns $e_\oplus, e_\otimes$ when l is false or true w.r.t. $\mathcal{I}$, therefore the statement is true.

- Case $\alpha = \alpha_1 + \alpha_2$:
  The algorithm nondeterministically returns $\textsc{Eval}_{\mathcal{R}}(\alpha_1, \mathcal{I})$ or $\textsc{Eval}_{\mathcal{R}}(\alpha_2, \mathcal{I})$. By the induction hypothesis, we know that the sum of all the execution paths of $\textsc{Eval}_{\mathcal{R}}(\alpha_i, \mathcal{I})$ is equal to $[\![\alpha_i]\!]_{\mathcal{R}}(\mathcal{I})$. Since we guess $i \in \{1, 2\}$ nondeterministically $\textsc{Eval}_{\mathcal{R}}(\alpha, \mathcal{I})$ has all the execution paths of $\textsc{Eval}_{\mathcal{R}}(\alpha_1, \mathcal{I})$ and $\textsc{Eval}_{\mathcal{R}}(\alpha_2, \mathcal{I})$ and therefore the sum of all values produced by execution paths of $\textsc{Eval}_{\mathcal{R}}(\alpha, \mathcal{I})$ is equal to
  $$[\![\alpha_2]\!]_{\mathcal{R}}(\mathcal{I}) \oplus [\![\alpha_2]\!]_{\mathcal{R}}(\mathcal{I}) = [\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}).$$

- Case $\alpha = \alpha_1 * \alpha_2$:
  The algorithm returns one of the results of the execution paths of $\textsc{Eval}_{\mathcal{R}}(\alpha_1, \mathcal{I})$ multiplied by one of the results of the execution paths of $\textsc{Eval}_{\mathcal{R}}(\alpha_2, \mathcal{I})$. By the induction hypothesis, we know that the sum of all the execution paths of $\textsc{Eval}_{\mathcal{R}}(\alpha_i, \mathcal{I})$ is equal to $[\![\alpha_i]\!]_{\mathcal{R}}(\mathcal{I})$. For $\textsc{Eval}_{\mathcal{R}}(\alpha, \mathcal{I})$, we have one nondeterministic execution path for every combination of nondeterministic execution paths of $\textsc{Eval}_{\mathcal{R}}(\alpha_i, \mathcal{I}), i = 1, 2$ since the nondeterministic choices are made independently. Therefore, if the results of the execution paths of $\textsc{Eval}_{\mathcal{R}}(\alpha_i, \mathcal{I})$ are $k_1^{(i)}, \ldots, k_{n_i}^{(i)}$ we see that the sum of all the execution paths of $\textsc{Eval}_{\mathcal{R}}(\alpha, \mathcal{I})$ is
  $$\bigoplus_{j_1=1}^{n_1} \bigoplus_{j_2=1}^{n_2} k_{j_1}^{(1)} \otimes k_{j_2}^{(1)}$$
  By using distributivity, we obtain that this is equal to
  $$\bigoplus_{j_1=1}^{n_1} \left(k_{j_1}^{(1)} \otimes \bigoplus_{j_2=1}^{n_2} k_{j_2}^{(1)} \right) = \bigoplus_{j_1=1}^{n_1} k_{j_1}^{(1)} \otimes \bigoplus_{j_2=1}^{n_2} k_{j_2}^{(1)} \overset{IH}{=} [\![\alpha_1]\!]_{\mathcal{R}}(\mathcal{I}) \otimes [\![\alpha_2]\!]_{\mathcal{R}}(\mathcal{I}) = [\![\alpha_1 * \alpha_2]\!]_{\mathcal{R}}(\mathcal{I}).$$

- Case $\alpha = \Sigma v\ \alpha$:
  Works analogously to case $\alpha = \alpha_1 + \alpha_2$.

□

B.2.2 Complexity of SAT($\mathcal{R}$), SumProd($\mathcal{R}$), AMC, Algebraic Measure Evaluation, SCSP, ΣFO-Eval($\mathcal{R}$), mrg(F), Datalog Semiring Provenance

↪ **Theorems 90, 93 to 96 and 100**

For the Karp-reducibility between SAT($\mathcal{R}$), $\textsc{SumProd}(\mathcal{R})$, AMC, Algebraic Measure Evaluation, SCSP, ΣFO-$\textsc{Eval}(\mathcal{R})$, mrg($F$), we do not prove interreducibility between two problems at a time but prove it by using the strategy visualized in Figure B.1.

Theorem B.2.2. *For every commutative semiring $\mathcal{R}$ it holds that each of the following problems can be reduced to one another using Karp s-reductions:*

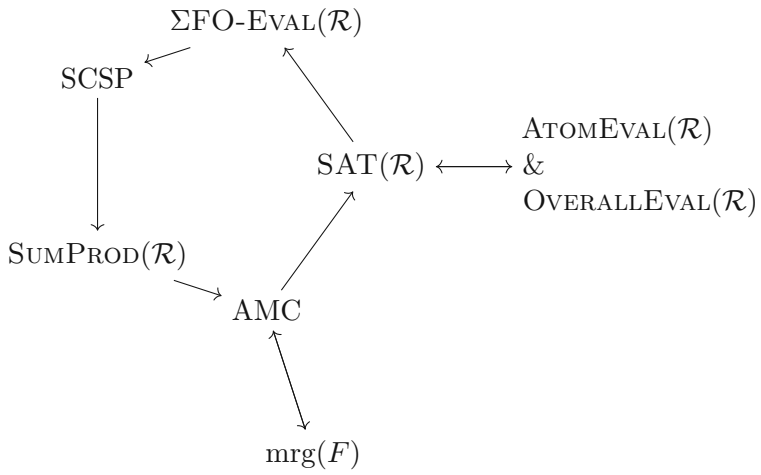


Figure B.1: Karp s-reductions that are proven to show Theorems 90, 93 to 96 and 100. $P \to Q$ means that a Karp s-reduction from P to Q is given.

- *SAT($\mathcal{R}$),*
- *ΣFO-Eval($\mathcal{R}$),*
- *Computing blevel(P) of an SCSP P over $\mathcal{R}$,*
- *SumProd($\mathcal{R}$),*
- *Algebraic measure querying over $\mathcal{R}$,*
- *AMC over $\mathcal{R}$, and*
- *Computing mrg(F) for a semiring-labeled CNF F over $\mathcal{R}$.*

Proof. Karp s-reducibility is transitive, therefore it suffices to prove

- SAT($\mathcal{R}$) is Karp s-reducible to ΣFO-Eval($\mathcal{R}$)
- ΣFO-Eval($\mathcal{R}$) is Karp s-reducible to computing *blevel*(P) of an SCSP P over $\mathcal{R}$
- Computing *blevel*(P) of an SCSP P over $\mathcal{R}$ is Karp s-reducible to SumProd($\mathcal{R}$)
- SumProd($\mathcal{R}$) is Karp s-reducible to AMC over $\mathcal{R}$
- AMC over $\mathcal{R}$ is Karp s-reducible to SAT($\mathcal{R}$)
- AMC over $\mathcal{R}$ is Karp s-reducible to mrg(F) over $\mathcal{R}$
- mrg(F) over $\mathcal{R}$ is Karp s-reducible to AMC over $\mathcal{R}$
- SAT($\mathcal{R}$) is Karp s-reducible to algebraic measure querying over $\mathcal{R}$

- AtomEval($\mathcal{R}$) and OverallEval($\mathcal{R}$) are Karp s-reducible to SAT($\mathcal{R}$)

□

This gives us that as desired

Corollary B.2.3. *Theorems 90, 93 to 96 and 100 hold.*

Proof. We proved in Theorem 88 that SAT($\mathcal{R}$) is NP($\mathcal{R}$)-complete. It follows from Theorem B.2.2 and Lemma 87 that also all other problems are NP($\mathcal{R}$)-complete. Additionally, Karp s-reducibility is shown in Theorem B.2.2. □

We proceed to prove the Karp s-reducibilities in the order specified in the above Theorem.

Lemma B.2.4. *SAT($\mathcal{R}$) is Karp s-reducible to ΣFO-Eval($\mathcal{R}$)*

Proof. Let α be a ΣBF over $\mathcal{R}$ with variable $v_1, \ldots, v_n$. We choose σ as the triple $\langle\{\bot, \top\}, \{t(.)\}, \{x_{v_1}, \ldots, x_{v_n}\}\rangle$ and $\mathcal{I} = \{t(\top), \neg t(\bot)\}$. Then we replace every propositional variable v in α by $t(x_v)$, which symbolizes that v has truth value $\top$, and every quantifier Σv with the corresponding first order quantifier Σx_v. For the resulting ΣFO formula β, it is easy to see that

$$[\![\alpha]\!]_{\mathcal{R}}(\emptyset) = [\![\beta]\!]_{\mathcal{R}}(\mathcal{I})$$

since the semantics are analogously defined. □

Lemma B.2.5. *ΣFO-Eval($\mathcal{R}$) is Karp s-reducible to computing blevel(P) of an SCSP P over $\mathcal{R}$*

This lemma is the one that contains the most difficult steps, since it requires reducing arbitrary combinations of sums and products to a sum of product.

We first prove some general structural assumptions that we can make about ΣFO formulas:

Lemma B.2.6. *For every ΣFO weighted formula α over a commutative semiring $\mathcal{R}$, domain $\mathcal{D}$ and interpretation $\mathcal{I}$ we can construct in polynomial time in $\alpha, \mathcal{D}, \mathcal{I}$ a ΣFO weighted formula $\beta = \Sigma x_1 \ldots \Sigma x_n \gamma$, where γ is quantifier free, over $\mathcal{R}$ and an interpretation $\mathcal{I}'$ such that*

$$[\![\alpha]\!]^{\sigma}_{\mathcal{R}}(\mathcal{I}) = [\![\beta]\!]^{\sigma}_{\mathcal{R}}(\mathcal{I}').$$

Proof. Let α be a ΣFO weighted formula over semiring $\mathcal{R}$, $\mathcal{D}$ be a domain and $\mathcal{I}$ be an interpretation. We show that we can always push out the sum quantifiers.
Let $\alpha = (\Sigma x \alpha_1) * \alpha_2$. W.l.o.g. we can assume that α_2 does not contain x. Then

$$[\![\alpha]\!]^{\sigma}_{\mathcal{R}}(\mathcal{I}) = [\![(\Sigma x \alpha_1) * \alpha_2]\!]^{\sigma}_{\mathcal{R}}(\mathcal{I})$$

$$= [\![\Sigma x \alpha_1]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) \otimes [\![\alpha_2]\!]^\sigma_{\mathcal{R}}(\mathcal{I})$$
$$= (\bigoplus_{d \in \mathcal{D}} [\![\alpha_1\{x \mapsto d\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I})) \otimes [\![\alpha_2]\!]^\sigma_{\mathcal{R}}(\mathcal{I})$$

Since addition distributes over multiplication and α_2 does not contain x, we get

$$\begin{aligned}[\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) &= \bigoplus_{d \in \mathcal{D}} ([\![\alpha_1\{x \mapsto d\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) \otimes [\![\alpha_2\{x \mapsto d\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) \\ &= \bigoplus_{d \in \mathcal{D}} [\![(\alpha_1 * \alpha_2)\{x \mapsto d\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) \\ &= [\![\Sigma x(\alpha_1 * \alpha_2)]\!]^\sigma_{\mathcal{R}}(\mathcal{I})\end{aligned}$$

So we can choose $\beta = \Sigma x(\alpha_1 * \alpha_2)$ and $\mathcal{I}' = \mathcal{I}$.
Let $\alpha = \alpha_1 * (\Sigma x \alpha_2)$. This case works analogously to the previous one.
Let $\alpha = (\Sigma x \alpha_1) + \alpha_2$. We choose some $d \in D$, a new predicate symbol p_{d^*} with arity one and define $\mathcal{I}' = \mathcal{I} \cup \{p_{d^*}(d^*)\}$. Then $\beta = \Sigma x(\alpha_1 + (\alpha_2 * p_{d^*}(x)))$ has the following semantics w.r.t $\mathcal{I}'$:

$$\begin{aligned}[\![\beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}') &= [\![\Sigma x(\alpha_1 + (\alpha_2 * p_{d^*}(x)))]\!]^\sigma_{\mathcal{R}}(\mathcal{I}') \\ &= \bigoplus_{d \in D} [\![(\alpha_1 + (\alpha_2 * p_{d^*}(x)))\{x \mapsto d\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}') \\ &= \bigoplus_{d \in D} [\![\alpha_1\{x \mapsto d\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}') \oplus [\![(\alpha_2 * p_{d^*}(x))\{x \mapsto d\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}') \\ &= \bigoplus_{d \in D} [\![\alpha_1\{x \mapsto d\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}') \oplus \begin{cases} [\![\alpha_2\{x \mapsto d\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}') & d = d^* \\ e_\oplus & \text{otherwise.} \end{cases}\end{aligned}$$

Since p_{d^*} is a new predicate that does not occur in α, we can remove it from $\mathcal{I}'$

$$\begin{aligned}[\![\beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}') &= \bigoplus_{d \in D} [\![\alpha_1\{x \mapsto d\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) \oplus \begin{cases} [\![\alpha_2]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) & d = d^* \\ e_\oplus & \text{otherwise.} \end{cases} \\ &= (\bigoplus_{d \in D} [\![\alpha_1\{x \mapsto d\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I})) \oplus [\![\alpha_2]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) \\ &= [\![\Sigma x \alpha_1]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) \oplus [\![\alpha_2]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) \\ &= [\![(\Sigma x \alpha_1) + \alpha_2]\!]^\sigma_{\mathcal{R}}(\mathcal{I})\end{aligned}$$

So we can choose $\beta = \Sigma x(\alpha_1 + (\alpha_2 * p_{d^*}(x)))$, $\mathcal{I}' = \mathcal{I} \cup \{p_{d^*}(d^*)\}$.
The case $\alpha = \alpha_1 + (\Sigma x \alpha_2)$ works analogously.
This shows that we can always push the Σ quantifier to the top in polynomially many steps in the number of occurrences of $*$ and $+$ in α. Since we only need to add one predicate p_{d^*} to handle all the disjunctions, the size of the interpretation only increases polynomially. Last but not least, we only add $*p_{d^*}(x)$ to the formula at most once for every combination of Σ and $+$ in α, the size of β is also polynomial in α. □

Lemma B.2.7. *For every ΣFO weighted formula α over a commutative semiring $\mathcal{R}$, finite domain $\mathcal{D}$, and interpretation $\mathcal{I}$, then we can construct in polynomial time a ΣFO weighted formula β over $\mathcal{R}$ such that in every subformula $\Sigma x \gamma$ the variable x occurs in γ and*

$$[\![\alpha]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) = [\![\beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}).$$

Proof. Let α be a ΣFO weighted formula over semiring $\mathcal{R}$, $\mathcal{D}$ be a domain, and $\mathcal{I}$ be an interpretation. We iteratively replace every subformula $\Sigma x\gamma$ where γ does not contain x of α by a formula with the same semantics w.r.t. $\mathcal{D}$ and $\mathcal{I}$.
So let $\Sigma x\gamma$ be a weighted formula where γ does not contain x. It has the semantics

$$\begin{aligned}
[\![\Sigma x\gamma]\!]^\sigma_\mathcal{R}(\mathcal{I}) &= \bigoplus_{d\in\mathcal{D}} [\![\gamma\{x\mapsto d\}]\!]^\sigma_\mathcal{R}(\mathcal{I}) \\
&= \bigoplus_{d\in\mathcal{D}} (e_\otimes \otimes [\![\gamma]\!]^\sigma_\mathcal{R}(\mathcal{I})) \\
&= (\bigoplus_{d\in\mathcal{D}} e_\otimes) \otimes [\![\gamma]\!]^\sigma_\mathcal{R}(\mathcal{I}) \\
&= (\bigoplus_{d\in\mathcal{D}} [\![e_\otimes]\!]^\sigma_\mathcal{R}(\mathcal{I})) \otimes [\![\gamma]\!]^\sigma_\mathcal{R}(\mathcal{I}) \\
&= [\![e_\otimes + \ldots + e_\otimes]\!]^\sigma_\mathcal{R}(\mathcal{I}) \otimes [\![\gamma]\!]^\sigma_\mathcal{R}(\mathcal{I}) \\
&= [\![(e_\otimes + \ldots + e_\otimes) * \gamma]\!]^\sigma_\mathcal{R}(\mathcal{I}).
\end{aligned}$$

So we can replace $\Sigma x\gamma$ by $(e_\otimes + \ldots + e_\otimes) * \gamma$ without changing the semantics of the formula. This is further possible in polynomial time and the resulting formula only increases in size polynomially in $|\mathcal{D}|$. This stays true, when we iteratively perform the replacement for all such quantifiers Σx. □

We need one more auxiliary lemma:

Lemma B.2.8. *For each clause $a_1 \vee a_2 \vee a_3$, where a_i is a literal over the variables $v_s, s \in S$ we can construct a constraint $c_{a_1\vee a_2\vee a_3} = \langle def, con\rangle$ s.t. $def(d_1, d_2, d_3) = e_\otimes$ if d_1, d_2, d_3 is a satisfying assignment of $a_1 \vee a_2 \vee a_3$.*

Proof. We can do this as follows.

$$def(d_1, d_2, d_3) = \begin{cases} & d_1, d_2, d_3 \in \{\bot, \top\}, \\ e_\otimes & \exists i : (a_i = v_s \wedge d_i = \top) \\ & \vee (a_i = \neg v_s \wedge d_i = \bot) \\ e_\oplus & \text{otherwise.} \end{cases} \tag{B.1}$$

con is simply the set of variables that occur in $a_1 \vee a_2 \vee a_3$. □

Proof of Lemma B.2.5. So let α be a ΣFO weighted formula over $\mathcal{R}$ and domain $\mathcal{D}$ that is to be evaluated using interpretation $\mathcal{I}$.

We use Lemmas B.2.6 and B.2.7 to ensure that α the input formula is of the form

$$\Sigma x_1 \ldots \Sigma x_n \beta$$

for some quantifier free β, that contains every variable $x_i, i = 1, \ldots, n$. We define $P = \langle C, \{s, x_1, \ldots, x_n\}\rangle$ inductively on the structure of β s.t.

$$(\Pi C) \Downarrow_{\{x_1,\ldots,x_n\}} = \langle def, \{x_1, \ldots, x_n\}\rangle,$$

and for all $(d_1, \ldots, d_n) \in \mathcal{D}^n$ it holds that

$$def(d_1, \ldots, d_n) = [\![\beta\{x_i \mapsto d_i\}]\!]^{\sigma}_{\mathcal{R}}(\mathcal{I})$$

whereas for all $(d_1, \ldots, d_n) \in D^n \setminus \mathcal{D}^n$ (remember that $\mathcal{D}$ is the domain of the ΣFO formula and D is the domain of the SCSP) it holds that

$$def'(d_1, \ldots, d_n) = e_{\oplus}.$$

Then

$$\begin{aligned}
blevel(P) &= (\Pi C) \Downarrow_{\emptyset} \\
&= (\Pi C) \Downarrow_{\{x_1,\ldots,x_n\}} \Downarrow_{\emptyset} \\
&= \langle def, \{x_1, \ldots, x_n\}\rangle \Downarrow_{\emptyset} \\
&= \bigoplus_{(d_1,\ldots,d_n)\in D^n} def'(d_1, \ldots, d_n) \\
&= \bigoplus_{(d_1,\ldots,d_n)\in \mathcal{D}^n} def'(d_1, \ldots, d_n) \\
&= \bigoplus_{(d_1,\ldots,d_n)\in \mathcal{D}^n} [\![\beta\{x_i \mapsto d_i\}]\!]^{\sigma}_{\mathcal{R}}(\mathcal{I}) \\
&= [\![\Sigma x_1 \ldots \Sigma x_n \beta]\!]^{\sigma}_{\mathcal{R}}(\mathcal{I}) \\
&= [\![\alpha]\!]^{\sigma}_{\mathcal{R}}(\mathcal{I})
\end{aligned}$$

So let $\mathcal{I}$ be an interpretation and $\mathcal{D}$ a domain. We first define the domain D of the constraint system as $D := \mathcal{D} \cup \{\top, \bot\}$

Definition B.2.9 (Subformulas)**.** *Let α a quantifier free weighted formula. Then $\mathcal{S}(\alpha)$ the set of subformulas of α indexed by position $p \in \{0,1\} \times \{.0,.1\}^*$ is $\mathcal{S}^0(\alpha)$, where $\mathcal{S}^i(\alpha)$ for $i = 0, 1$ is defined using induction on the structure of the formula as follows:*

- *Case $\alpha \in \{p(\vec{x}), \neg p(\vec{x}), k\}$: Then $\mathcal{S}^i(\alpha) = \{(i, \alpha)\}$ for $i = 0, 1$.*
- *Case $\alpha = \alpha_1 + \alpha_2$ or $\alpha = \alpha_1 * \alpha_2$: Then $\mathcal{S}^i(\alpha) = \{(i, \alpha)\} \cup \{(i.r, \beta) \mid (r, \beta) \in \mathcal{S}^0(\alpha_1)\} \cup \{(i.r, \beta) \mid (r, \beta) \in \mathcal{S}^1(\alpha_2)\}$ for $i = 0, 1$.*

We assert (in the following referred to as (*)) that for values $\vec{d} \in D^k$, for which we do not fix the value $def(\vec{d})$, it is zero, i.e., $def(\vec{d}) = e_{\oplus}$. We define P using $\mathcal{S}(\beta)$. Intuitively, the strategy is as follows: For each indexed subformula of (s, γ) we add a variable v_s that determines whether the subformula should be included in the computation. Then for atomic subformulas we add a constraint that has the value of the atomic formula if it is included. For complex subformula $\alpha_1 * \alpha_2$, resp. $\alpha_1 + \alpha_2$ we add constraints that ensure that if the subformula should be included, then also α_1 and α_2, resp. α_1 xor α_2 have to be included.

Formally, we add the following constraints:

- For each $(s,k) \in \mathcal{S}(\beta)$ s.t. $k \in R$:
  Add the constraint that evaluates to k when the variable v_s has value $\top$ and evaluates to $e_\otimes$ otherwise. We add the constraint set

$$\begin{aligned} C_{(s,k)} =& \{c_{(s,k)}\} \\ c_{(s,k)} =& \langle \{\top \mapsto k, \bot \mapsto e_\otimes\}, \{v_s\} \rangle \end{aligned}$$

- For each $(s, p(\vec{x})) \in \mathcal{S}(\beta)$:
  We can see $p(\vec{x})$ as a constraint on the variables in $\vec{x}$ that evaluates to $e_\otimes$ when $p(\sigma(\vec{x})) \in \mathcal{I}$ for the assignment σ of the variables $\mathrm{var}(\vec{x})$ in $\vec{x}$ and $e_\oplus$ otherwise. This however only happens when the variable v_s has value $\top$, otherwise the value is $e_\otimes$. As a consequence we add the constraint set

$$\begin{aligned} C_{(s,p(\vec{x}))} &= \{c_{(s,p(\vec{x}))}\} \\ c_{(s,p(\vec{x}))} &= \langle \{(\top, d_1, \ldots, d_k) \mapsto e_\otimes \mid p(\vec{x})\{x_i \mapsto d_i\} \in \mathcal{I}, d_i \in \mathcal{D}\} \\ &\quad \cup \{(\top, d_1, \ldots, d_k) \mapsto e_\oplus \mid p(\vec{x})\{x_i \mapsto d_i\} \notin \mathcal{I}, d_i \in \mathcal{D}\} \\ &\quad \cup \{(\bot, d_1, \ldots, d_k) \mapsto e_\otimes \mid d_i \in \mathcal{D}\}, \\ &\quad \{v_s\} \cup \mathrm{var}(\vec{x}) \rangle \end{aligned}$$

- For each $(s, \neg p(\vec{x})) \in \mathcal{S}(\beta)$:
  Analogously with $e_\otimes$ and $e_\oplus$ swapped. We add the constraint set

$$\begin{aligned} C_{(s,\neg p(\vec{x}))} &= \{c_{(s,\neg p(\vec{x}))}\} \\ c_{(s,\neg p(\vec{x}))} &= \langle \{(\top, d_1, \ldots, d_k) \mapsto e_\oplus \mid p(\vec{x})\{x_i \mapsto d_i\} \in \mathcal{I}, d_i \in \mathcal{D}\} \\ &\quad \cup \{(\top, d_1, \ldots, d_k) \mapsto e_\otimes \mid p(\vec{x})\{x_i \mapsto d_i\} \notin \mathcal{I}, d_i \in \mathcal{D}\} \\ &\quad \cup \{(\bot, d_1, \ldots, d_k) \mapsto e_\otimes \mid d_i \in \mathcal{D}\}, \\ &\quad \{v_s\} \cup \mathrm{var}(\vec{x}) \rangle \end{aligned}$$

- For each $(s, \alpha) \in \mathcal{S}(\beta)$, $\alpha = \alpha_1 * \alpha_2$:
  Then we know that $(s.0, \alpha_1), (s.1, \alpha_2) \in \mathcal{S}(\beta)$. We require that for $v_s, v_{s.0}, v_{s.1}$ the following relation holds:

$$\begin{aligned} v_s &\to (v_{s.0} \wedge v_{s.1}) \\ \neg v_s &\to (\neg v_{s.0} \wedge \neg v_{s.1}) \end{aligned}$$

  We can rewrite them in 3CNF as follows

$$\begin{aligned} &(\neg v_s \vee v_{s.0}) \wedge (\neg v_s \vee v_{s.1}) \\ &(v_s \vee \neg v_{s.0}) \wedge (v_s \vee \neg v_{s.1}). \end{aligned}$$

  Then using Lemma B.2.8 (where $a_2 = a_3$ is possible) we add the constraint set

$$C_{(s,\alpha)} = \{c_{\neg v_s \vee v_{s.0}}, c_{\neg v_s \vee v_{s.1}}, c_{v_s \vee \neg v_{s.0}}, c_{v_s \vee \neg v_{s.1}}\}.$$

- For each $(s, \alpha) \in \mathcal{S}(\beta)$, $\alpha = \alpha_1 + \alpha_2$:
  Then we know that $(s.0, \alpha_1), (s.1, \alpha_2) \in \mathcal{S}(\beta)$. We require that for $v_s, v_{s.0}, v_{s.1}$ the following relation holds:

$$v_s \to (v_{s.0} \wedge \neg v_{s.1} \vee \neg v_{s.0} \wedge v_{s.1})$$
$$\neg v_s \to (\neg v_{s.0} \wedge \neg v_{s.1})$$

  We can rewrite them in 3CNF as follows

$$(\neg v_s \vee v_{s.0} \vee v_{s.1}) \wedge (\neg v_s \vee \neg v_{s.0} \vee \neg v_{s.1})$$
$$(v_s \vee \neg v_{s.0}) \wedge (v_s \vee \neg v_{s.1}).$$

  Then using Lemma B.2.8 we add the constraint set

$$C_{(s,\alpha)} = \{c_{\neg v_s \vee v_{s.0} \vee v_{s.1}}, c_{\neg v_s \vee \neg v_{s.0} \vee \neg v_{s.1}}, c_{v_s \vee \neg v_{s.0}}, c_{v_s \vee \neg v_{s.1}}\}.$$

- We further need to assert that s_0, i.e. the variable specifying whether the root node will be included in the computation, is true: We add it again using Lemma B.2.8 (with $a_1 = a_2 = a_3 = v_0$)

$$c_{v_0}.$$

So overall the constraint problem P is given as

$$\langle \bigcup_{(s,\alpha)\in\mathcal{S}(\beta)} C_{(s,\alpha)} \cup \{c_{v_0}\}, \{v_s \mid (s, \alpha) \in \mathcal{S}(\beta)\} \cup \{x_1, \ldots, x_n\}\rangle.$$

This is feasible to construct in polynomial time in the size of β.

We prove the correctness of the approach by structural induction on β. We use the induction invariant that for

$$(\Pi \bigcup_{(s,\alpha)\in\mathcal{S}(\beta)} C_{(s,\alpha)}) \Downarrow_{\{v_0,x_1,\ldots,x_n\}} = \langle def, \{v_0, x_1, \ldots, x_n\}\rangle,$$

and for all $d_1, \ldots, d_n \in \mathcal{D}$

$$def(d_0, \ldots, d_n) = \begin{cases} [\![\beta\{x_i \mapsto d_i\}]\!]^{\sigma}_{\mathcal{R}}(\mathcal{I}) & d_0 = \top \\ e_{\otimes} & d_0 = \bot \\ e_{\oplus} & \text{otherwise.} \end{cases}$$

holds.

- $\beta = k$, $k \in R$:
  Then $P = \langle\langle\{\top \mapsto k\} \cup \{\bot \mapsto e_{\otimes}\}, \{v_0\}\rangle, \{v_0\}\rangle$. We know

$$(\Pi C) \Downarrow_{v_0} = \langle\langle\{\top \mapsto k, \bot \mapsto e_{\otimes}\}, \{v_0\}\rangle.$$

Then due to Assertion (*)

$$def(d_0) = \begin{cases} k & d = \top \\ e_\otimes & d = \bot \\ e_\oplus & \text{otherwise.} \end{cases} = \begin{cases} [\![\beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) & d_0 = \top \\ e_\otimes & d_0 = \bot \\ e_\oplus & \text{otherwise.} \end{cases}$$

- $\beta = p(\vec{x})$:
  Then

$$\begin{aligned} P = \langle \{ \langle \{ (\top, d_1, \ldots, d_k) \mapsto e_\otimes \mid p(\vec{x})\{x_i \mapsto d_i\} \in \mathcal{I}, d_i \in \mathcal{D}\} \\ \cup \{ (\top, d_1, \ldots, d_k) \mapsto e_\oplus \mid p(\vec{x})\{x_i \mapsto d_i\} \notin \mathcal{I}, d_i \in \mathcal{D}\} \\ \cup \{ (\bot, d_1, \ldots, d_k) \mapsto e_\otimes \mid d_i \in \mathcal{D}\}, \\ \{v_0\} \cup \mathrm{var}(\vec{x}) \rangle \} \cup \{c_{v_0}\}, \\ \{v_0\} \cup \mathrm{var}(\vec{x}) \rangle. \end{aligned}$$

  Let

$$(\Pi C) \Downarrow_{\{v_0\} \cup \mathrm{var}(\vec{x})} = \langle def, \{v_0, x_1, \ldots, x_n\} \rangle,$$

  and $d_1, \ldots, d_n \in \mathcal{D}$. Then

$$def(d_0, d_1, \ldots, d_n) = \begin{cases} e_\otimes & d_0 = \top, p(\vec{x})\{x_i \mapsto d_i\} \in \mathcal{I} \\ e_\oplus & d_0 = \top, p(\vec{x})\{x_i \mapsto d_i\} \notin \mathcal{I} \\ e_\otimes & d_0 = \bot, \\ e_\oplus & \text{otherwise.} \end{cases} = \begin{cases} [\![\beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) & d_0 = \top \\ e_\otimes & d_0 = \bot \\ e_\oplus & \text{otherwise.} \end{cases}$$

- $\beta = \neg p(\vec{x})$:
  Then

$$\begin{aligned} P = \langle \{ \langle \{ (\top, d_1, \ldots, d_k) \mapsto e_\otimes \mid p(\vec{x})\{x_i \mapsto d_i\} \notin \mathcal{I}, d_i \in \mathcal{D}\} \\ \cup \{ (\top, d_1, \ldots, d_k) \mapsto e_\oplus \mid p(\vec{x})\{x_i \mapsto d_i\} \in \mathcal{I}, d_i \in \mathcal{D}\} \\ \cup \{ (\bot, d_1, \ldots, d_k) \mapsto e_\otimes \mid d_i \in \mathcal{D}\}, \\ \{v_0\} \cup \mathrm{var}(\vec{x}) \rangle \} \cup \{c_{v_0}\}, \\ \{v_0\} \cup \mathrm{var}(\vec{x}) \rangle. \end{aligned}$$

  Let

$$(\Pi C) \Downarrow_{\{v_0\} \cup \mathrm{var}(\vec{x})} = \langle def, \{v_0, x_1, \ldots, x_n\} \rangle,$$

and $d_1, \ldots, d_n \in \mathcal{D}$. Then

$$def(d_0, d_1, \ldots, d_n) = \bigoplus_{d \in D} \begin{cases} e_\otimes & d_0 = \top, p(\vec{x})\{x_i \mapsto d_i\} \notin \mathcal{I} \\ e_\oplus & d_0 = \top, p(\vec{x})\{x_i \mapsto d_i\} \in \mathcal{I} \\ e_\otimes & d_0 = \bot, \\ e_\oplus & \text{otherwise.} \end{cases}$$
$$= \begin{cases} [\![\beta]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) & d_0 = \top \\ e_\otimes & d_0 = \bot \\ e_\oplus & \text{otherwise.} \end{cases}$$

- $\beta = \beta_1 * \beta_2$:
  Then

$$\langle \bigcup_{(s,\alpha) \in \mathcal{S}(\beta)} C_{(s,\alpha)}, \{v_s \mid (s,\alpha) \in \mathcal{S}(\beta)\} \cup \{x_1, \ldots, x_n\} \rangle$$
$$= \langle \bigcup_{(s,\alpha) \in \mathcal{S}^0(\beta_1)} C_{(0.s,\alpha)} \cup \bigcup_{(s,\alpha) \in \mathcal{S}^1(\beta_2)} C_{(1.s,\alpha)} \cup C_{(0,\beta)},$$
$$\{v_{0.s} \mid (s,\alpha) \in \mathcal{S}^0(\beta_1)\} \cup \{v_{1.s} \mid (s,\alpha) \in \mathcal{S}^1(\beta_2)\} \cup \{v_0\} \cup \mathrm{var}(\beta_1) \cup \mathrm{var}(\beta_2) \rangle$$

  Now if we let the constraint problems defined for β_1, β_2 be $P_i = \langle C_i, con_i \rangle$, we can see that (modulo appropriate renaming of the variables of the form v_s)

$$\langle \bigcup_{(s,\alpha) \in \mathcal{S}(\beta)} C_{(s,\alpha)}, \{v_s \mid (s,\alpha) \in \mathcal{S}(\alpha)\} \cup \{x_1, \ldots, x_n\} \rangle$$
$$= \langle C_1 \cup C_2 \cup C_{(0,\beta)},$$
$$con_1 \cup con_2 \cup \{v_0\} \rangle$$

  Using the induction hypothesis we get that

$$(\Pi C_i) \Downarrow_{\{v_{0.i}, \mathrm{var}(\beta_i)\}} = \langle def_i, \{v_{0.i}\} \cup \mathrm{var}(\beta_i) \rangle,$$

  and for $d_1, \ldots, d_n \in \mathcal{D}$

$$def_i(d_0, \ldots, d_{n_i}) = \begin{cases} [\![\beta_i\{x_i \mapsto d_i\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) & d_0 = \top \\ e_\otimes & d_0 = \bot \\ e_\oplus & \text{otherwise.} \end{cases}$$

  Then for

$$(\Pi C) \Downarrow_{\{v_0, \mathrm{var}(\beta)\}} = \langle def, \{v_0\} \cup \mathrm{var}(\beta) \rangle,$$

  we see that

$$def(d_0, \ldots, d_n)$$
$$= \bigoplus_{\vec{d} \in D^{con_1 \cup con_2}} \bigotimes_{\langle def_c, con_c \rangle \in C_1 \cup C_2 \cup C_{(0,\beta)}} def_c(\vec{d} \downarrow^{con}_{con_c})$$

$$= \bigoplus_{\vec{d}\in D^{con_1\cup con_2}} \bigotimes_{\langle def_c, con_c\rangle\in C_1\cup C_2} def_c(\vec{d}\downarrow^{con}_{con_c}) \otimes \begin{cases} & d_0, d_{v_{0.0}}, d_{v_{0.1}} \in \{\bot,\top\}, \\ e_\otimes & d_0 \to (d_{v_{0.0}} \wedge d_{v_{0.1}}) \\ & \wedge \neg d_0 \to (\neg d_{v_{0.0}} \wedge \neg d_{v_{0.1}}) \\ e_\oplus & \text{otherwise.} \end{cases}$$

$$= \bigoplus_{\vec{d}\in D^{con_1\cup con_2}} \begin{cases} \bigotimes_{\langle def_c, con_c\rangle\in C_1\cup C_2} def_c(\vec{d}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} = d_{v_{0.1}} = \top, \\ \bigotimes_{\langle def_c, con_c\rangle\in C_1\cup C_2} def_c(\vec{d}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} = d_{v_{0.1}} = \bot, \\ e_\oplus & \text{otherwise.} \end{cases}$$

$$= \bigoplus_{\vec{d}\in D^{con_1\cup con_2}} \begin{cases} \bigotimes_{\langle def_c, con_c\rangle\in C_1} def_c(\vec{d}\downarrow^{con}_{con_c}) & \\ \otimes\bigotimes_{\langle def_c, con_c\rangle\in C_2} def_c(\vec{d}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} = d_{v_{0.1}} = \top, \\ \bigotimes_{\langle def_c, con_c\rangle\in C_1} def_c(\vec{d}\downarrow^{con}_{con_c}) & \\ \otimes\bigotimes_{\langle def_c, con_c\rangle\in C_2} def_c(\vec{d}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} = d_{v_{0.1}} = \bot, \\ e_\oplus & \text{otherwise.} \end{cases}$$

$$= \bigoplus_{\vec{d_1}\in D^{con_1}} \bigoplus_{\vec{d_2}\in D^{con_2}} \begin{cases} \bigotimes_{\langle def_c, con_c\rangle\in C_1} def_c(\vec{d_1}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} \\ & \| \\ \otimes\bigotimes_{\langle def_c, con_c\rangle\in C_2} def_c(\vec{d_2}\downarrow^{con}_{con_c}) & d_{v_{0.1}} = \top, \\ \bigotimes_{\langle def_c, con_c\rangle\in C_1} def_c(\vec{d_1}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} \\ & \| \\ \otimes\bigotimes_{\langle def_c, con_c\rangle\in C_2} def_c(\vec{d_2}\downarrow^{con}_{con_c}) & d_{v_{0.1}} = \bot, \\ e_\oplus & \text{otherwise.} \end{cases}$$

$$= \bigoplus_{d_{v_{0.0}}, d_{v_{0.1}}\in D} \begin{cases} \bigoplus_{\vec{d_1}\in D^{con_1}\setminus\{v_{0.0}\}} \bigotimes_{\langle def_c, con_c\rangle\in C_1} def_c(\vec{d_1}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} \\ & \| \\ \otimes \bigoplus_{\vec{d_2}\in D^{con_2}\setminus\{v_{0.1}\}} \bigotimes_{\langle def_c, con_c\rangle\in C_2} def_c(\vec{d_2}\downarrow^{con}_{con_c}) & d_{v_{0.1}} = \top, \\ \bigoplus_{\vec{d_1}\in D^{con_1}\setminus\{v_{0.0}\}} \bigotimes_{\langle def_c, con_c\rangle\in C_1} def_c(\vec{d_1}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} \\ & \| \\ \otimes \bigoplus_{\vec{d_2}\in D^{con_2}\setminus\{v_{0.1}\}} \bigotimes_{\langle def_c, con_c\rangle\in C_2} def_c(\vec{d_2}\downarrow^{con}_{con_c}) & d_{v_{0.1}} = \bot, \\ e_\oplus & \text{otherwise.} \end{cases}$$

$$= \begin{cases} def_1(\top, d_1, \ldots, d_{n_1}) * def_2(\top, d_1, \ldots, d_{n_2}) & d_0 = \top, \\ def_1(\bot, d_1, \ldots, d_{n_1}) * def_2(\bot, d_1, \ldots, d_{n_2}) & d_0 = \bot, \\ e_\oplus & \text{otherwise.} \end{cases}$$

$$= \begin{cases} [\![\beta_1\{x_i \mapsto d_i\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) \otimes [\![\beta_2\{x_i \mapsto d_i\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) & d_0 = \top, \\ e_\otimes & d_0 = \bot, \\ e_\oplus & \text{otherwise.} \end{cases}$$

$$= \begin{cases} [\![(\beta_1 * \beta_2)\{x_i \mapsto d_i\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) & d_0 = \top, \\ e_\otimes & d_0 = \bot, \\ e_\oplus & \text{otherwise.} \end{cases}$$

- $\beta = \beta_1 + \beta_2$:
  Then

$$\langle \bigcup_{(s,\alpha)\in\mathcal{S}(\beta)} C_{(s,\alpha)}, \{v_s \mid (s,\alpha) \in \mathcal{S}(\beta)\} \cup \{x_1, \ldots, x_n\}\rangle$$

$$= \langle \bigcup_{(s,\alpha)\in\mathcal{S}^0(\beta_1)} C_{(0.s,\alpha)} \cup \bigcup_{(s,\alpha)\in\mathcal{S}^1(\beta_2)} C_{(1.s,\alpha)} \cup C_{(0,\beta)},$$
$$\{v_{0.s} \mid (s,\alpha) \in \mathcal{S}^0(\beta_1)\} \cup \{v_{1.s} \mid (s,\alpha) \in \mathcal{S}^1(\beta_2)\} \cup \{v_0\} \cup \mathrm{var}(\beta_1) \cup \mathrm{var}(\beta_2)\rangle$$

Now if we let the constraint problems defined for β_1, β_2 be $P_i = \langle C_i, con_i\rangle$, we can see that (modulo appropriate renaming of the variables of the form v_s)

$$\begin{aligned}&\langle \bigcup_{(s,\alpha)\in\mathcal{S}(\beta)} C_{(s,\alpha)}, \{v_s \mid (s,\alpha) \in \mathcal{S}(\alpha)\} \cup \{x_1,\ldots,x_n\}\rangle\\ =\ &\langle C_1 \cup C_2 \cup C_{(0,\beta)},\\ &\ con_1 \cup con_2 \cup \{v_0\}\rangle\end{aligned}$$

Using the induction hypothesis we get that

$$(\Pi C_i) \Downarrow_{\{v_{0.i},\mathrm{var}(\beta_i)\}} = \langle def_i, \{v_{0.i}\} \cup \mathrm{var}(\beta_i)\rangle,$$

and for $d_1,\ldots,d_n \in \mathcal{D}$

$$def_i(d_0,\ldots,d_{n_i}) = \begin{cases} [\![\beta_i\{x_i \mapsto d_i\}]\!]^{\sigma}_{\mathcal{R}}(\mathcal{I}) & d_0 = \top \\ e_{\otimes} & d_0 = \bot \\ e_{\oplus} & \text{otherwise.}\end{cases}$$

Then

$$\begin{aligned}&def(d_0,\ldots,d_n)\\ =\ &\bigoplus_{\vec{d}\in D^{con_1\cup con_2}} \bigotimes_{\langle def_c,con_c\rangle\in C_1\cup C_2\cup C_{(0,\beta)}} def_c(\vec{d}\downarrow^{con}_{con_c})\\ =\ &\bigoplus_{\vec{d}\in D^{con_1\cup con_2}} \bigotimes_{\langle def_c,con_c\rangle\in C_1\cup C_2} def_c(\vec{d}\downarrow^{con}_{con_c}) \otimes \begin{cases} & d_0, d_{v_{0.0}}, d_{v_{0.1}} \in \{\bot,\top\},\\ e_{\otimes} & d_0 \to (d_{v_{0.0}} \text{ xor } d_{v_{0.1}})\\ & \wedge \neg d_0 \to (\neg d_{v_{0.0}} \wedge \neg d_{v_{0.1}})\\ e_{\oplus} & \text{otherwise.}\end{cases}\\ =\ &\bigoplus_{\vec{d}\in D^{con_1\cup con_2}} \begin{cases} \bigotimes_{\langle def_c,con_c\rangle\in C_1\cup C_2} def_c(\vec{d}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.1}} = \top, d_{v_{0.0}} = \bot,\\ \bigotimes_{\langle def_c,con_c\rangle\in C_1\cup C_2} def_c(\vec{d}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} = \top, d_{v_{0.1}} = \bot,\\ \bigotimes_{\langle def_c,con_c\rangle\in C_1\cup C_2} def_c(\vec{d}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} = d_{v_{0.1}} = \bot,\\ e_{\oplus} & \text{otherwise.}\end{cases}\\ =\ &\bigoplus_{\vec{d}_1\in D^{con_1}} \bigoplus_{\vec{d}_2\in D^{con_2}} \begin{cases} \bigotimes_{\langle def_c,con_c\rangle\in C_1} def_c(\vec{d}_1\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.1}} = \top\\ \otimes\bigotimes_{\langle def_c,con_c\rangle\in C_2} def_c(\vec{d}_2\downarrow^{con}_{con_c}) & \text{and } d_{v_{0.0}} = \bot,\\ \bigotimes_{\langle def_c,con_c\rangle\in C_1} def_c(\vec{d}_1\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} = \top\\ \otimes\bigotimes_{\langle def_c,con_c\rangle\in C_2} def_c(\vec{d}_2\downarrow^{con}_{con_c}) & \text{and } d_{v_{0.1}} = \bot,\\ \bigotimes_{\langle def_c,con_c\rangle\in C_1} def_c(\vec{d}_1\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}}\\ & \quad\| \\ \otimes\bigotimes_{\langle def_c,con_c\rangle\in C_2} def_c(\vec{d}_2\downarrow^{con}_{con_c}) & d_{v_{0.1}} = \bot,\\ e_{\oplus} & \text{otherwise.}\end{cases}\end{aligned}$$

$$= \bigoplus_{d_{v_{0.0}}, d_{v_{0.1}} \in D} \begin{cases} \bigoplus_{\vec{d_1} \in D^{con_1} \setminus \{v_{0.0}\}} \bigotimes_{\langle def_c, con_c \rangle \in C_1} def_c(\vec{d_1}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.1}} = \top \\ \otimes \bigoplus_{\vec{d_2} \in D^{con_2} \setminus \{v_{0.1}\}} \bigotimes_{\langle def_c, con_c \rangle \in C_2} def_c(\vec{d_2}\downarrow^{con}_{con_c}) & \text{and } d_{v_{0.0}} = \bot, \\ \bigoplus_{\vec{d_1} \in D^{con_1} \setminus \{v_{0.0}\}} \bigotimes_{\langle def_c, con_c \rangle \in C_1} def_c(\vec{d_1}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} = \top \\ \otimes \bigoplus_{\vec{d_2} \in D^{con_2} \setminus \{v_{0.1}\}} \bigotimes_{\langle def_c, con_c \rangle \in C_2} def_c(\vec{d_2}\downarrow^{con}_{con_c}) & \text{and } d_{v_{0.1}} = \bot, \\ \bigoplus_{\vec{d_1} \in D^{con_1} \setminus \{v_{0.0}\}} \bigotimes_{\langle def_c, con_c \rangle \in C_1} def_c(\vec{d_1}\downarrow^{con}_{con_c}) & d_0 = d_{v_{0.0}} \\ & \quad \| \\ \otimes \bigoplus_{\vec{d_2} \in D^{con_2} \setminus \{v_{0.1}\}} \bigotimes_{\langle def_c, con_c \rangle \in C_2} def_c(\vec{d_2}\downarrow^{con}_{con_c}) & d_{v_{0.1}} = \bot, \\ e_\oplus & \text{otherwise.} \end{cases}$$

$$= \begin{cases} def_1(\top, d_1, \ldots, d_{n_1}) \otimes def_2(\bot, d_1, \ldots, d_{n_2}) & \\ \oplus def_1(\bot, d_1, \ldots, d_{n_1}) \otimes def_2(\top, d_1, \ldots, d_{n_2}) & d_0 = \top, \\ def_1(\bot, d_1, \ldots, d_{n_1}) \otimes def_2(\bot, d_1, \ldots, d_{n_2}) & d_0 = \bot, \\ e_\oplus & \text{otherwise.} \end{cases}$$

$$= \begin{cases} [\![\beta_1\{x_i \mapsto d_i\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) \otimes e_\otimes \oplus e_\otimes \otimes [\![\beta_2\{x_i \mapsto d_i\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) & d_0 = \top, \\ e_\otimes & d_0 = \bot, \\ e_\oplus & \text{otherwise.} \end{cases}$$

$$= \begin{cases} [\![(\beta_1 + \beta_2)\{x_i \mapsto d_i\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) & d_0 = \top, \\ e_\otimes & d_0 = \bot, \\ e_\oplus & \text{otherwise.} \end{cases}$$

Now that we know that the induction invariant holds, i.e., for

$$(\Pi \bigcup_{(s,\alpha)\in\mathcal{S}(\beta)} C_{(s,\alpha)}) \Downarrow_{\{v_0, x_1, \ldots, x_n\}} = \langle def, \{v_0, x_1, \ldots, x_n\}\rangle,$$

and $d_1, \ldots, d_n \in \mathcal{D}$

$$def(d_0, \ldots, d_n) = \begin{cases} [\![\beta\{x_i \mapsto d_i\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) & d_0 = \top \\ e_\otimes & d_0 = \bot \\ e_\oplus & \text{otherwise.} \end{cases}$$

it remains to show that for

$$(\Pi \bigcup_{(s,\alpha)\in\mathcal{S}(\beta)} C_{(s,\alpha)} \cup \{c_{v_0}\}) \Downarrow_{\{x_1, \ldots, x_n\}} = \langle def', \{x_1, \ldots, x_n\}\rangle,$$

and for all $d_1, \ldots, d_n \in \mathcal{D}$

$$def'(d_1, \ldots, d_n) = [\![\beta\{x_i \mapsto d_i\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}).$$

This is because

$$\begin{aligned} def'(d_1, \ldots, d_n) &= \bigoplus_{d_0 \in D} def(d_0, \ldots, d_n) \otimes \bigotimes_{\langle def_c, con_c \rangle \in \{c_{v_0}\}} def_c(\vec{d}\downarrow^{con}_{con_c}) \\ &= \bigoplus_{d_0 \in D} \begin{cases} [\![\beta\{x_i \mapsto d_i\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) & d_0 = \top \\ e_\otimes & d_0 = \bot \\ e_\oplus & \text{otherwise.} \end{cases} \otimes \begin{cases} e_\otimes & d_0 = \top, \\ e_\oplus & \text{otherwise.} \end{cases} \\ &= \bigoplus_{d_0 \in D} \begin{cases} [\![\beta\{x_i \mapsto d_i\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}) & d_0 = \top \\ e_\oplus & \text{otherwise.} \end{cases} \\ &= [\![\beta\{x_i \mapsto d_i\}]\!]^\sigma_{\mathcal{R}}(\mathcal{I}). \end{aligned}$$

□

Lemma B.2.10. *Computing blevel(P) of an SCSP P over $\mathcal{R}$ is Karp s-reducible to the problem SumProd$(\mathcal{R})$.*

Proof. Let $P = \langle C, con\rangle$ be some SCSP over $\langle \mathcal{R}, D, V\rangle$. Recall that

$$\begin{aligned} blevel(P) &= (\Pi_{\langle def, con'\rangle \in C} \langle def, con'\rangle) \Downarrow_{\emptyset} \\ &= \bigoplus_{\vec{d} \in D^{con}} \bigotimes_{\langle def, con'\rangle \in C} def(\vec{d}\downarrow^{con}_{con'}) \end{aligned}$$

We therefore choose the SumProd$(\mathcal{R})$-instance with variables $X_1, \ldots, X_n$, where $n = |con|$ and functions $\{def \mid \langle def, con\rangle \in C\}$. Then the value of this SumProd$(\mathcal{R})$-instance is equal to $blevel(P)$. □

Lemma B.2.11. *SumProd$(\mathcal{R})$ is Karp s-reducible to AMC over $\mathcal{R}$*

Proof. Assume we are given a SumProd$(\mathcal{R})$-instance over variables $X_1, \ldots, X_n$, functions $f_1, \ldots, f_m$ with inputs $\vec{Y_1}, \ldots, \vec{Y_m}$ and domain $\mathcal{D}$. We construct the AMC-instance as follows.

We add the variables

- $v_{i,d}$ for each $i = 1, \ldots, n$ and $d \in D$, where $v_{i,j}$ is true if X_i takes values d
- $v_{j,\vec{d}}$ for each $j = 1, \ldots, m$ and $d \in D^{Y_j}$, where $v_{j,\vec{d}}$ is true if the input to the function f_j is equal to $\vec{d}$

We define α by setting

- $\alpha(v_{i,d}) = \alpha(\neg v_{i,d}) = e_{\otimes}$
- $\alpha(v_{j,\vec{d}}) = f_j(\vec{d})$ and $\alpha(\neg v_{j,\vec{d}}) = e_{\otimes}$

We define T as the propositional theory containing the following formulas

- $\neg v_{i,d} \vee \neg v_{i,d'}$ for each $i = 1, \ldots, n$ and $d \neq d' \in D$
- $\bigvee_{d \in D} v_{i,d}$ for each $i = 1, \ldots, n$
- $v_{j,\vec{d}} \leftrightarrow \bigwedge_{X_i \in \vec{Y_j}} v_{i,\vec{d}_{X_i}}$ for each $j = 1, \ldots, m$ and $\vec{d} \in D^{\vec{Y_j}}$

Then every satisfying interpretation $\mathcal{I}$ of T corresponds to one assignment of the variables X_i to domain values d_i. Furthermore, for each such $\mathcal{I}$ the variables $v_{j,\vec{d}}$ tell us which input the function f_j gets.

Since we sum up $\bigotimes_{v\in\mathcal{I}}\alpha(v)$ for each satisfying interpretation $\mathcal{I}$ and for each such $\mathcal{I}$ it holds that $\bigotimes_{j=1}^{m}\alpha(v_{j,\vec{d}}) = \bigotimes_{j=1}^{m} f_j(\vec{d})$, we see that $A(T)$ over $\mathcal{R}$ is exactly the value of the SumProd($\mathcal{R}$)-instance. □

Lemma B.2.12. *AMC over $\mathcal{R}$ is Karp s-reducible to SAT($\mathcal{R}$).*

Proof. Let the AMC-instance be over variables $v_1, \ldots, v_n$, theory T and weight function α.

W.l.o.g. we can assume that the theory T consists of a single propositional formula ϕ (otherwise we simply take their conjunction). We apply the Tseitin transformation [Tse83] to ϕ and obtain a 3CNF ψ with additional variables $x_1, \ldots, x_m$ s.t. each satisfying assignment of ϕ is uniquely extendable to a satisfying assignment of ψ. Furthermore, each satisfying assignment of ψ is a satisfying assignment of ϕ when restricted to the variables $v_1, \ldots, v_n$.

Recall that by Lemma 106, for each clause $l_1 \vee l_2 \vee l_3$ it holds that

$$\begin{aligned}\mathcal{I} \models l_1 \vee l_2 \vee l_3 &\iff [\![l_1 + \neg l_1 * l_2 + \neg l_1 * \neg l_2 * l_3]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\otimes}\\ \mathcal{I} \not\models l_1 \vee l_2 \vee l_3 &\iff [\![l_1 + \neg l_1 * l_2 + \neg l_1 * \neg l_2 * l_3]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\oplus}.\end{aligned}$$

So we can proceed as follows: in ψ we replace every clause $l_1 \vee l_2 \vee l_3$ by $l_1 + \neg l_1 * l_2 + \neg l_1 * \neg l_2 * l_3$ and subsequently every $\wedge$ by $*$. We call the resulting formula β and see that

$$\begin{aligned}\mathcal{I} \models \psi &\iff [\![\beta]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\otimes}\\ \mathcal{I} \not\models \psi &\iff [\![\beta]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\oplus}\end{aligned}$$

Therefore, the ΣBF γ defined as

$$\Sigma v_1 \ldots \Sigma v_n \Sigma x_1 \ldots \Sigma x_m \beta * \Pi_{i=1}^{n}(v_i * \alpha(v_i) + \neg v_i * \alpha(\neg v_i))$$

fulfills

$$A(T) = [\![\gamma]\!]_{\mathcal{R}}(\emptyset).$$

□

Lemma B.2.13. *AMC over $\mathcal{R}$ is Karp s-reducible to mrg(F) over $\mathcal{R}$ and vice versa.*

Proof. We first reduce AMC over $\mathcal{R}$ to mrg(F) over $\mathcal{R}$. Assume we are given an AMC-instance (T, α) over a commutative semiring $\mathcal{R}$ and variables $\mathcal{V}$. W.l.o.g., we can assume

that T is a CNF $C_1 \wedge \cdots \wedge C_n$. If this was not the case, we could apply the Tseitin-transformation and extend α to the added variables by letting $\alpha(l) = e_\otimes$ for each of their literals l. We construct a semiring-weighted CNF F as

$$\begin{aligned} F &= F_T \cup F_+ \cup F_-, \text{ where} \\ F_T &= \{(C_1, e_\oplus), \ldots, (C_n, e_\oplus)\}, \\ F_+ &= \{(\neg v, \alpha(v)) \mid v \in \mathcal{V}\}, \\ F_- &= \{(v, \alpha(\neg v)) \mid v \in \mathcal{V}\}. \end{aligned}$$

Then

$$\begin{aligned} mrg(F) &= \bigoplus_{\mathcal{I} \in \mathrm{Int}(F)} \phi_F(\mathcal{I}) \\ &= \bigoplus_{\mathcal{I} \models T} \phi_{F_+ \cup F_-}(\mathcal{I}) \\ &= \bigoplus_{\mathcal{I} \models T} \bigotimes_{v \in V} \left\{ \begin{array}{cc} \alpha(v) & \mathcal{I} \not\models \neg v \\ e_\otimes & \text{otherwise.} \end{array} \right\} \otimes \bigotimes_{v \in V} \left\{ \begin{array}{cc} \alpha(\neg v) & \mathcal{I} \not\models v \\ e_\otimes & \text{otherwise.} \end{array} \right\} \\ &= \bigoplus_{\mathcal{I} \models T} \bigotimes_{v \in V} \left\{ \begin{array}{cc} \alpha(v) & \mathcal{I} \models v \\ e_\otimes & \text{otherwise.} \end{array} \right\} \otimes \bigotimes_{v \in V} \left\{ \begin{array}{cc} \alpha(\neg v) & \mathcal{I} \models \neg v \\ e_\otimes & \text{otherwise.} \end{array} \right\} \\ &= \bigoplus_{\mathcal{I} \models T} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \notin \mathcal{I}} \alpha(\neg v) \\ &= A(T). \end{aligned}$$

We see that the result of the marginalization problem of F over $\mathcal{R}$ is equal to $A(T)$, the algebraic model count of the instance (T, α). Since we can construct F in polynomial time from (T, α), this shows that the AMC over $\mathcal{R}$ is Karp s-reducible to mrg(F) over $\mathcal{R}$.

Next, we reduce mrg(F) over $\mathcal{R}$ to AMC over $\mathcal{R}$. Assume we are given a semiring-labeled CNF $F = \{(C_1, w_1), \ldots, (C_n, w_n)\}$. Consider the following AMC instance with theory T and labeling function α such that

$$\begin{aligned} T &= \bigwedge_{i=1}^{n} C_i \leftrightarrow c_i \\ \alpha(l) &= \left\{ \begin{array}{ll} w_i & \text{if } l = \neg c_i, \\ e_\otimes & \text{otherwise.} \end{array} \right. , \end{aligned}$$

where $c_1, \ldots, c_n$ are distinct variables that do not occur in F. Then

$$\begin{aligned} A(T) &= \bigoplus_{\mathcal{I} \models T} \bigotimes_{v \in \mathcal{I}} \alpha(v) \otimes \bigotimes_{v \notin \mathcal{I}} \alpha(\neg v) \\ &= \bigoplus_{\mathcal{I} \models T} \bigotimes_{c_i \notin \mathcal{I}} \alpha(\neg c_i) \\ &= \bigoplus_{\mathcal{I}} \bigotimes_{i, \text{ s.t. } \mathcal{I} \not\models C_i} w_i \\ &= mrg(F). \end{aligned}$$

Since we can construct (T, α) in polynomial time from F, this shows that the mrg(F) over $\mathcal{R}$ is Karp s-reducible to AMC over $\mathcal{R}$. □

Lemma B.2.14. *OverallEval($\mathcal{R}$) is Karp s-reducible to SAT($\mathcal{R}$) and vice versa.*

Proof. We first note that we can Karp s-reduce OverallEval($\mathcal{R}$) and AtomEval($\mathcal{R}$) to one another. For this, consider an algebraic measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$. Then the overall weight query $\mu(\Pi)$ has the same result as the query $\mu'(a)$ for the atom a that does not occur in Π, where μ' is the measure $\langle \Pi \cup \{a \leftarrow\}, \alpha, \mathcal{R} \rangle$. On the other hand, the query $\mu(a)$ for an atom $a \in \mathcal{A}(\Pi)$ has the same result as the overall weight query $\mu'(\Pi \cup \{\leftarrow a\})$, where $\mu' = \langle \Pi \cup \{\leftarrow a\}, \alpha, \mathcal{R} \rangle$. We can, thus, proceed by proving the claim for overall weight queries and obtain the same result for queries for an atom.

We first reduce SAT($\mathcal{R}$) to OverallEval($\mathcal{R}$). So, w.l.o.g., let α be a ΣBF formula of the form $\alpha = \Sigma a_1 \Sigma a_2 \dots \Sigma a_n \beta$, where β is Σ-free. Consider the algebraic measure $\mu = \langle \Pi, \beta, \mathcal{R} \rangle$, where $\Pi = \{\{a_i\} \leftarrow | \ i = 1, \dots, n\}$. Then the answer sets of Π are equal to the subsets of $\{a_1, \dots, a_n\}$ and accordingly

$$\mu(\Pi) = \bigoplus_{\mathcal{I} \subseteq \{a_1, \dots, a_n\}} [\![\beta]\!]_{\mathcal{R}}(\mathcal{I}) = [\![\alpha]\!]_{\mathcal{R}}(\emptyset).$$

For the other direction, consider an algebraic measure $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$. It is a well known fact [JN11] that for Π there exists a propositional theory T with auxiliary variables from a set X such that there is a bijection f from the answer sets of Π to Models(T), the set of satisfying assignments of T, such that for each answer set $\mathcal{I} \in \mathcal{AS}(\Pi)$ it holds that $f(\mathcal{I}) \cap \mathcal{A}(\Pi) = \mathcal{I}$. I.e., the answer set $\mathcal{I}$ and the satisfying assignment $f(\mathcal{I})$ agree on the original variables. Thus, it holds that

$$\begin{aligned} \mu(\Pi) &= \bigoplus_{\mathcal{I} \in \mathcal{AS}(\Pi)} [\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}) \\ &= \bigoplus_{\mathcal{I} \in \mathcal{AS}(\Pi)} [\![\alpha]\!]_{\mathcal{R}}(f(\mathcal{I})) \\ &= \bigoplus_{\mathcal{I} \in \text{Models}(T)} [\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}), \end{aligned}$$

where the second equality is due to the fact that the auxiliary variables do not occur in X.

We use the fact we showed in the proof of Lemma B.2.12 that for a propositional theory T there exists a weighted formula β using auxiliary variables from a set Y such that there is a bijection f from satisfying assignments of T to the interpretations $\mathcal{I}$ of β with $[\![\beta]\!]_{\mathcal{R}}(\mathcal{I}) \neq e_{\oplus}$, such that for each model $\mathcal{I} \in \text{Models}(T)$ it holds that $[\![\beta]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\otimes}$ and $f(\mathcal{I}) \cap \mathcal{A}(T) = \mathcal{I}$, where $\mathcal{A}(T)$ denotes the set of propositional variables in T.

Thus, we obtain the following equalities:

$$\begin{aligned} \mu(\Pi) &= \bigoplus_{\mathcal{I} \in \text{Models}(T)} [\![\alpha]\!]_{\mathcal{R}}(\mathcal{I}) \\ &= \bigoplus_{\mathcal{I} \in \text{Models}(T)} [\![\beta * \alpha]\!]_{\mathcal{R}}(f(\mathcal{I})) \\ &= \bigoplus_{\mathcal{I} \subseteq \mathcal{A}(T) \cup Y} [\![\beta * \alpha]\!]_{\mathcal{R}}(\mathcal{I}) \\ &= [\![\Sigma a_1 \dots \Sigma a_n \Sigma x_1 \dots \Sigma x_m \Sigma y_1 \dots \Sigma y_k \beta * \alpha]\!]_{\mathcal{R}}(\emptyset), \end{aligned}$$

where $\{a_1, \dots, a_n\} = \mathcal{A}(\Pi), X = \{x_1, \dots, x_m\}$ and $Y = \{y_1, \dots, y_k\}$. Since the latter is a ΣBF formula, we are done. □

B.3 Relation to classical complexity classes

B.3.1 FPSpace(poly)-membership of SAT($\mathcal{R}$) for efficiently encoded semirings

↪ **Proposition 103**

Proposition 103 (FPSpace(poly) Upper-Bound)**.** *If $e(\mathcal{R})$ is an efficiently encoded commutative semiring, then SAT($e(\mathcal{R})$) is in FPSpace(poly).*

Proof. We have seen that SAT($e(\mathcal{R})$) is Karp s-reducible to SumProd($e(\mathcal{R})$), therefore it is sufficient to prove FPSpace(poly)-membership for SumProd($e(\mathcal{R})$).

Assume we are given a SumProd($e(\mathcal{R})$)-instance and a domain $\mathcal{D}$ as a set of variables $X_1, \ldots, X_n$ and functions $f_1, \ldots, f_m$ with input vectors $\vec{Y_1}, \ldots, \vec{Y_m}$ consisting of variables from $X_1, \ldots, X_n$. The value of this instance is then given by

$$a = \bigoplus_{x_1,\ldots,x_n \in \mathcal{D}} \bigotimes_{i=1}^{m} f_i(\vec{y_i}).$$

We can bound $\|a\|_e$ by using that $e(\mathcal{R})$ is efficiently encoded.

$$\begin{aligned}
\|a\|_e &= \|\bigoplus_{x_1,\ldots,x_n \in \mathcal{D}} \bigotimes_{i=1}^{m} f_i(\vec{y_i})\|_e \\
&\leq p(\log_2 |\mathcal{D}|^n) \max_{x_1,\ldots,x_n \in \mathcal{D}} \|\bigotimes_{i=1}^{m} f_i(\vec{y_i})\|_e \\
&\leq p(n \log_2 |\mathcal{D}|) \max_{x_1,\ldots,x_n \in \mathcal{D}} p(m) \max_{i=1,\ldots,m} \|f_i(\vec{y_i})\|_e \\
&\leq p(n \log_2 |\mathcal{D}|) p(m) \max_{x_1,\ldots,x_n \in \mathcal{D}} \max_{i=1,\ldots,m} \|f_i(\vec{y_i})\|_e
\end{aligned}$$

Therefore, the size of the result a and consequently also all intermediate results are bounded polynomially in the size of the input. We see that we at least have enough space to store all the results and intermediate results. Furthermore, we know that multiplication and addition are in FP and therefore also in FPSpace(poly). Last but not least, iterating over all the assignments of values $d_i \in \mathcal{D}$ to variables X_i is also possible in polynomial space. □

B.3.2 NP, #P, GapP, OptP-completeness of SAT($\mathbb{B}$), SAT($\mathbb{N}$), SAT($\mathbb{Z}$), SAT($\mathcal{R}_{\max,+}$)

↪ **Theorem 104**

Theorem 104. *For $(\mathcal{R}, \mathcal{C}) = (\mathbb{B}, NP)$, $(\mathbb{N}, \#P)$, $(\mathbb{Z}, GapP)$, $(\mathcal{R}_{\max,+}, OptP)$ and the binary representation bin of the integers, SAT(bin($\mathcal{R}$)) is $\mathcal{C}$-complete w.r.t. Karp reductions.*

Proof. Membership is not hard to see:

- SAT(bin($\mathbb{B}$)) is SAT;

- SAT(bin($\mathbb{N}$)) can be solved in #P by simulating Algorithm 1, where instead of returning k we generate k accepting paths;
- SAT(bin($\mathbb{Z}$)) can be solved in GapP by simulating Algorithm 1, where instead of returning k we generate k accepting and zero rejecting paths if $k \geq 0$ and zero accepting and $|k|$ rejecting paths, otherwise;
- SAT(bin($\mathcal{R}_{\max,+}$)) can be solved in OptP by simulating Algorithm 1, exactly as it is.

.

For $\mathcal{C}$-hardness, we consider the following reductions

- SAT(bin($\mathbb{B}$)) is SAT;
- We can reduce #SAT to SAT(bin($\mathbb{N}$)) as follows.

  Let ϕ be the propositional formula, whose satisfying assignments we want to count. We apply the Tseitin transformation [Tse83] to ϕ and obtain a 3CNF ψ with additional variables $x_1, \ldots, x_m$ s.t. each satisfying assignment of ϕ is uniquely extendable to a satisfying assignment of ψ. Furthermore, for each satisfying assignment of ψ is a satisfying assignment of ϕ when restricted to the original variables $v_1, \ldots, v_n$.

  Recall that by Lemma 106, for each clause $l_1 \vee l_2 \vee l_3$ it holds that

$$\begin{aligned} \mathcal{I} \models l_1 \vee l_2 \vee l_3 &\iff [\![l_1 + \neg l_1 * l_2 + \neg l_1 * \neg l_2 * l_3]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\otimes} \\ \mathcal{I} \not\models l_1 \vee l_2 \vee l_3 &\iff [\![l_1 + \neg l_1 * l_2 + \neg l_1 * \neg l_2 * l_3]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\oplus}. \end{aligned}$$

  So we can proceed as follows: in ψ we replace every clause $l_1 \vee l_2 \vee l_3$ by $l_1 + \neg l_1 * l_2 + \neg l_1 * \neg l_2 * l_3$ and subsequently every $\wedge$ by $*$. The resulting formula β fulfills

$$\begin{aligned} \mathcal{I} \models \psi &\iff [\![\beta]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\otimes} \\ \mathcal{I} \not\models \psi &\iff [\![\beta]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\oplus} \end{aligned}$$

  Therefore, the semantics of the ΣBF defined as

$$\Sigma v_1 \ldots \Sigma v_n \Sigma x_1 \ldots \Sigma x_m \beta$$

  is exactly the number of satisfying assignments of ϕ.

- Recall that for an $n \times n$ matrix A with entries a_{ij},the permanent is given by

$$\sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i\sigma(i)},$$

where S_n is the set of permutations of the numbers $1, \ldots, n$. We reduce computing the permanent of a given integer matrix A, which is well-known to be #P-complete [Val79], to SAT(bin($\mathbb{Z}$)) as follows.

We use variables v_{ij}, which are true when we include the value a_{ij} in the current product. We construct a weighted QBF α s.t. $[\![\alpha]\!]_{\mathbb{Z}}(\mathcal{I}) = e_{\otimes}$ if the variables $v_{ij} \in \mathcal{I}$ correspond to a permutation and $e_{\oplus}$ otherwise.

We define α as the product of the following weighted formulas

- $\neg v_{ij} + v_{ij} * \neg v_{ij'}$ for each $i = 1, \ldots, n$ and $j \neq j' = 1, \ldots, n$ (include at most one element per row)
- $v_{i1} + \ldots + v_{in}$ for each $i = 1, \ldots, n$ (include at least one element per column)
- $\neg v_{ij} + v_{ij} * \neg v_{i'j}$ for each $i \neq i' = 1, \ldots, n$ and $j = 1, \ldots, n$ (include at most one element per column)

Then the semantics of the ΣBF

$$\Sigma a_{11} \ldots \Sigma a_{nn} \alpha * \Pi_{i,j=1}^{n} (v_{ij} * a_{ij} + \neg v_{ij})$$

is exactly the permanent of A.

- We can reduce LexMaxSAT to SAT(bin($\mathcal{R}_{\max,+}$)) as follows. Let ϕ be the propositional formula whose maximum satisfying assignments we want to know. We again as in the proof of #P-hardness for SAT(bin($\mathbb{N}$)) obtain the formula β s.t.

$$\mathcal{I} \models \psi \iff [\![\beta]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\otimes}$$
$$\mathcal{I} \not\models \psi \iff [\![\beta]\!]_{\mathcal{R}}(\mathcal{I}) = e_{\oplus}$$

Then the semantics of the ΣBF

$$\Sigma v_1 \ldots \Sigma v_n \Sigma x_1 \ldots \Sigma x_m \beta * \Pi_{i=1}^{n} (v_i * 2^{n-i} + \neg v_i)$$

is the bitstring representing the maximum satisfying assignment of ϕ respectively $-\infty$ if there is none.

□

B.3.3 Results for classes of semirings

Reducibility via epimorphisms
↪ Theorem 111

Theorem 111. *Let $e_i(\mathcal{R}_i), i = 1, 2$ be two encoded commutative semirings, such that*

1. *there exists a polynomial time computable epimorphism $f : e_1(R_1) \to e_2(R_2)$, and*

2. *for each* $e_2(r_2) \in e(R_2)$ *one can compute in polynomial time* $e_1(r_1)$ *s.t.* $f(e_1(r_1)) = e_2(r_2)$ *from* $e_2(r_2)$.

Then SAT$(e_2(\mathcal{R}_2))$ *is counting-reducible to SAT*$(e_1(\mathcal{R}_1))$.

Proof. Let the assumptions of the theorem be given. Furthermore, let α be a SAT$(e_2(\mathcal{R}))$-instance. We can compute $[\![\alpha]\!]_{e_2(\mathcal{R})}(\emptyset)$ as follows.

First we replace every occurrence of a value $e_2(r_2)$ with a value $e_1(r_1)$ s.t. $f(e_1(r_1)) = e_2(r_2)$. This is possible in polynomial time. The resulting ΣBF β is a SAT$(e_1(\mathcal{R}_1))$-instance. We now compute $[\![\beta]\!]_{e_1(\mathcal{R}_1)}(\emptyset)$. Then we only need to apply f to the result and have the solution $[\![\alpha]\!]_{e_2(\mathcal{R})}(\emptyset)$. This is possible in polynomial time with one oracle call to SAT$(e_1(\mathcal{R}_1))$.

Since we only need to solve one SAT$(e_1(\mathcal{R}_1))$-instance and do not require any information about the original instance to obtain the final solution from the solution of the SAT$(e_1(\mathcal{R}_1))$-instance we have a counting-reduction. □

Hardest Semirings
↪ **Lemma 112**

Lemma 112 (Hardest Semirings)**.** *There exists an encoding* e^* *for*

1. $\mathbb{N}_{\leq o}[(x_i)_\infty]$;
2. $\mathbb{Z}_p[(x_i)_\infty]$;
3. $\mathbb{N}_{\leq o} \times \mathbb{Z}_p[(x_i)_\infty]$;
4. $\mathbb{N}[(x_i)_\infty]$

such that for any commutative efficiently encoded commutative semiring $e(\mathcal{R})$ *that is in addition*

1. *periodic with periodicity* 1;
2. *periodic with periodicity* $p \geq 2$ *and offset* 0;
3. *periodic with periodicity* $p \geq 2$ *and offset* $o > 0$;
4. *not periodic*

it holds that SAT$(e(\mathcal{R}))$ *is counting reducible to*

1. *SAT*$(e^*(\mathbb{N}_{\leq o}[(x_i)_\infty]))$;

2. *SAT($e^*(\mathbb{Z}_p[(x_i)_\infty])$);*

3. *SAT($e^*(\mathbb{N}_{\leq o} \times \mathbb{Z}_p[(x_i)_\infty])$);*

4. *SAT($e^*(\mathbb{N}[(x_i)_\infty])$), respectively.*

Proof. As the encoding function e^* we take the function that encodes a monomial by representing the coefficient and index of each variable in binary and each exponent in unary. Furthermore, a polynomial is encoded as the list of the encodings of its monomials with non-zero coefficients. Then we can define an epimorphism f by letting

$$f\left(\sum_{\vec{i}\in\mathbb{N}^*} a_{\vec{i}} x_0^{i_0} \cdot \dots \cdot x_{|\vec{i}|}^{i_{|\vec{i}|}}\right) := \bigoplus_{\vec{i}\in\mathbb{N}^*, i_j\neq 0 \Rightarrow \mathrm{bin}(i_j)\in e(R)} a_{\vec{i}} \cdot \left(\mathrm{bin}(0)^{i_0} \otimes \dots \otimes \mathrm{bin}(|\vec{i}|)^{i_{|\vec{i}|}}\right).$$

For this definition to make sense, recall that e maps to $\{0,1\}^*$ and therefore $e(R) \subseteq \{0,1\}^*$. Since we can assume w.l.o.g. that for all values $r \in R$ it holds that $e(r) \in \{0,1\}^*$ has a 1 as the first letter, it follows that for some $i_j \in \mathbb{N}$ it holds that $\mathrm{bin}(i_j) \in e(R)$. Therefore, the right-hand side of the definition is always a value in $e(R)$.

The idea of this definition is that we identify x_{i_j} with the value that $\mathrm{bin}(i_j)$ represents in $e(\mathcal{R})$. Then we can perform any calculation over $e^*(\mathcal{S}[(x_i)_\infty])$ with $\mathcal{S} = \mathbb{N}_{\leq o}, \mathbb{Z}_p, \mathbb{N}_{\leq o} \times \mathbb{Z}_p, \mathbb{N}$, depending on the periodicity and offset of $\mathcal{R}$, by using the variables as placeholders for the actual values. Finally, we can obtain the actual value over $e(R)$ using f. Note, that the sum only considers exponent vectors $\vec{i} \in \mathbb{N}^*$ where any exponent i_j of x_j is equal to zero, when $\mathrm{bin}(i_j)$ is not in $e(R)$ to ensure that f is well defined.

The check whether $\mathrm{bin}(i_j)$ is in $e(R)$ may be expensive. Therefore, f is not necessarily polynomial time computable over the whole semiring $e^*(\mathcal{S}[(x_i)_\infty])$. We consider instead $e^*(\mathcal{S}')$, where $\mathcal{S}' = \mathcal{S}[\{x_i \mid \mathrm{bin}(i) \in e(R)\}]$. Then, evaluating f over $e^*(\mathcal{S}')$ takes polynomial time in the size of the encoding of the input since $e(\mathcal{R})$ is efficiently encoded.

It is easy to see that the second condition of Theorem 111 is satisfied, since we can map $\mathrm{bin}(n) \in e(R)$ to $e^*(x_n)$, which is obviously contained in $e^*(\mathcal{S}')$.

Thus, we can apply Theorem 111 to show that $\mathrm{SAT}(e(\mathcal{R}))$ is counting reducible to $\mathrm{SAT}(e^*(\mathcal{S}'))$. Since $e^*(S')$ is a subset of $e^*(\mathcal{S}[(x_i)_\infty]$ it follows that $\mathrm{SAT}(e(\mathcal{S}'))$ is counting reducible to $\mathrm{SAT}(e^*(\mathcal{S}[(x_i)_\infty]))$. Since counting reducibility is transitive, we are done. □

Impossibility results for Polynomial Semirings
↪ Theorems 113, 114, 116 and 117

Theorem 113. *Let $\mathcal{R} = \mathbb{N}[(x_i)_\infty]$ (resp. $\mathcal{R} = \mathbb{B}[(x_i)_\infty]$). If there is an encoding function e for $\mathcal{R}$ s.t.*

1) $\|[\![\alpha]\!]_{\mathcal{R}}(\emptyset)\|_e$ is polynomial in the size of α,

2) we can extract the binary representation of the coefficient $n \in \mathbb{N}$ (resp. $b \in \mathbb{B}$) of $x_{i_1}^{j_1} ... x_{i_m}^{j_m}$ from $e(r)$ in time polynomial in $\|r\|_e$, and

3) $\|x_i\|_e$ is polynomial in i,

then $\#P \subseteq FP/\text{poly}$ (resp. $NP \subseteq P/\text{poly}$).

For the proof we use the following theorem due to Cadoli, Donini, and Schaerf [CDS96].

Theorem B.3.15. *Let Π be an NP-complete problem, and let $\Pi = \bigcup_{n\in\mathbb{N}} \Pi_n$, where $\Pi_n = \{\pi \in \Pi \mid |\pi| = n\}$. Moreover, let $[P, F, V]$ be a problem we want to compile, divided into fixed part F and varying part V. Suppose that there exists a polynomial p such that, for each $n > 0$, there exists an $f_n \in F$ with the following properties:*

1. *$|f_n| < p(n)$;*
2. *for all $\pi \in \Pi_n$, there exists a $v_\pi \in V$ such that:*
   a) *v_π can be computed from π in polynomial time;*
   b) *$\langle f_n, v_\pi \rangle$ is a "yes" instance of P iff π is a "yes" instance of Π.*

With the above hypothesis, if $[P, F, V]$ is compilable, then $NP \subseteq P/\text{poly}$.

The proof can be found in [CDS96] and can be extended to a proof for an analogous statement resulting in $\#\text{P} \subseteq \text{FP/poly}$.

Proof of Theorem 113. We use Theorem B.3.15 with the following $[P, F, V]$: P, the problem we consider, is checking for a monomial $V = x_{i_1}^{j_1} ... x_{i_n}^{j_n}$ whether the corresponding coefficient in $[\![\alpha]\!]_{\mathbb{N}[(x_i)_\infty]}(\emptyset)$ is unequal to zero. The fixed part F is given by the ΣBF α over the semiring $e(\mathbb{N}[(x_i)_\infty])$. We assume the encoding e of the polynomials is one that satisfies the precondition of Theorem 113.

We want to show that $[P, F, V]$ can be used to solve a #P-complete problem Π as in the precondition of Theorem B.3.15. As Π we choose #3SAT. We know that when for $\pi \in \Pi$ it holds that $|\pi| \leq n$ then π can contain at most n different variables. W.l.o.g. we can assume that the variables that are used are $a_1, \ldots, a_n$. Let $C_1, \ldots, C_k$ be all the three literal clauses constructible from $a_1, \ldots, a_n$; clearly k is polynomial in n.

We choose $f_n = \alpha_n$.

$$\alpha_n = \Sigma c_1 \ldots \Sigma c_k \Sigma a_1 \ldots \Sigma a_n \Pi_{i=1}^{k} (C_i' * c_i * e(x_i) + \neg c_i)$$

where $C_i = l_1 \vee l_2 \vee l_3$ is replaced by $C_i' = l_1 + \neg l_1 * l_2 + \neg l_1 * \neg l_2 * l_3$. Since k is polynomial in n and $\|x_i\|_e$ is polynomial in i, we know that α_n and therefore f_n is of polynomial size in n.

Given $\pi \in \Pi$ with $|\pi_n|$ we know that all the clauses are three literal clauses using only the variables $a_1, \ldots, a_n$. W.l.o.g. we can assume that every variable is used at least once, otherwise we can simply add $a_i \vee a_i \vee a_i$ for each variable that is not used without changing the number of satisfying assignments. Therefore, the set $C(\pi)$ of clauses in π is a subset of the clauses $C_1, \ldots, C_k$, i.e., $C(\pi) = \{C_{i_1}, \ldots, C_{i_m}\}$ where $1 \leq i_j < i_{j+1} \leq k$ for $j = 1, \ldots, m-1$. Then the coefficient of $x_{i_1} \cdot \cdots \cdot x_{i_m}$ in $[\![\alpha_n]\!]_{\mathbb{N}[(x_i)_\infty]}(\emptyset)$, which is

$$[\![\Sigma a_1 \ldots \Sigma a_n \Pi_{j=1}^m C'_{i_j}]\!]_{\mathbb{N}[(x_i)_\infty]}(\emptyset),$$

is equal to the number of satisfying assignments of π. Since we can extract the binary representation of the coefficient in polynomial time from the encoded value $e([\![\alpha_n]\!]_{\mathbb{N}[(x_i)_\infty]}(\emptyset))$, this would imply that $\#\mathrm{P} \subseteq \mathrm{FP/poly}$.

For $\mathbb{B}[(x_i)_\infty]$ we use the same setting, except that we use 3SAT and then the coefficient of $x_{i_1} \cdot \cdots \cdot x_{i_m}$ is equal to whether the 3SAT-instance is satisfiable. □

Theorem 114. *Let $\mathcal{R} = \mathbb{N}[x]$ (resp. $\mathcal{R} = \mathbb{B}[x]$). If there is an encoding function e for $\mathcal{R}$ s.t.*

1) *$\|[\![\alpha]\!]_{\mathcal{R}}(\emptyset)\|_e$ is polynomial in the size of α,*
2) *we can extract the binary representation of the coefficient $n \in \mathbb{N}$ (resp. $b \in \mathbb{B}$) of x^i from $e(r)$ in time polynomial in $\|r\|_e$, and*
3) *$\|x^i\|_e$ is polynomial in $\log_2(i)$,*

then $\#P \subseteq FP/\mathrm{poly}$ (resp. $NP \subseteq P/\mathrm{poly}$).

Proof. We proceed as in the proof of Theorem 113, this time however we use $f_n = \alpha_n$ where

$$\begin{aligned} \alpha_n =& \Sigma c_1 \ldots \Sigma c_k \Sigma a_1 \ldots \Sigma a_n \Pi_{i=1}^k C'_i * c_i * e(x^{2^i}) + \neg c_i, \\ v_n =& x^{2^{i_1} + \cdots + 2^{i_m}}. \end{aligned}$$

□

Theorem 115. *If $\#P \subseteq FP/\mathrm{poly}$ (resp. $NP \subseteq P/\mathrm{poly}$), then there exist encodings e_∞ and e_1 for $\mathbb{N}[(x_i)_\infty]$ and $\mathbb{N}[x]$ (resp. $\mathbb{B}[(x_i)_\infty]$ and $\mathbb{B}[x]$) such that the preconditions 1) - 3) of Theorems 113 and 114 are satisfied.*

Proof (sketch). For simplicity, we only consider the case where $\#\mathrm{P} \subseteq \mathrm{FP/poly}$ and argue for the existence of an encoding e_∞ for $\mathbb{N}[(x_i)_\infty]$ that satisfies the preconditions of Theorem 113. The proofs for the other cases use similar ideas.

The idea is as follows: First, we choose a basic encoding e of $\mathbb{N}[(x_i)_\infty]$, where (i) coefficients are encoded in binary, (ii) exponents are encoded in binary, (iii) variables indices are

encoded in unary, and (iv) monomials are encoded as tuples consisting of their coefficient plus a list of the variables together with their exponents. Finally, (v) polynomials are encoded as lists of their monomials with non-zero coefficients.

Clearly, this encoding does not satisfy condition 1). However, importantly it does satisfy condition 3). Before we continue, recall that given a $\mathrm{SAT}(e(\mathbb{N}[(x_i)_\infty]))$-instance α and a monomial $e(x_{i_1}^{j_1}...x_{i_n}^{j_n})$, it is possible to compute the coefficient of the monomial in $[\![\alpha]\!]_{e(\mathbb{N}[(x_i)_\infty])}(\emptyset)$ in #P. Assuming that $\#\mathrm{P} \subseteq \mathrm{FP/poly}$, it is also possible in FP/poly. Then, let A be an advice oracle that returns the polynomial size advice string $A(n)$ that can be used to solve any such coefficient query of input size n. Now we consider the representation of a polynomial p in $\mathbb{N}[(x_i)_\infty]$ as

$$(\beta, A(|(\beta, maxmon(\beta))|), A(|(\beta, maxmon(\beta))| - 1), \ldots, A(0)),$$

where

1. β is a $\mathrm{SAT}(e(\mathbb{N}[(x_i)_\infty]))$-instance such that $[\![\beta]\!]_{e(\mathbb{N}[(x_i)_\infty])}(\emptyset) = e(p)$,
2. $maxmon(\beta)$ is the monomial of maximum size encoding that has a non-zero coefficient in $[\![\beta]\!]_{\mathbb{N}[(x_i)_\infty]}(\emptyset)$, and
3. $|(\beta, maxmon(\beta))|$ denotes the size of the string that encodes the pair.

The size of the representation is polynomial in $\|\beta\|_e$, since

- $maxmon(\beta)$ is polynomial in $\|\beta\|_e$
- $|A(n)|$ is polynomial in n and, therefore,
- $|A(n)|$ is polynomial in $\|\beta\|_e$ for $n \leq |(\beta, maxmon(\beta))|$ and, therefore, also
- all $|(\beta, maxmon(\beta))|$ advice strings $A(i)$ concatenated are polynomial in $\|\beta\|_e$.

Consequently, there is a representation of the solution $[\![\alpha]\!]_{e(\mathbb{N}[(x_i)_\infty])}(\emptyset)$ of any instance α of $\mathrm{SAT}(e(\mathbb{N}[(x_i)_\infty]))$ whose size is polynomial in the size of α. Namely we can simply take

$$(\alpha, A(|(\alpha, maxmon(\alpha))|), A(|(\alpha, maxmon(\alpha))| - 1), \ldots, A(0)).$$

Furthermore, given this representation of $[\![\alpha]\!]_{e(\mathbb{N}[(x_i)_\infty])}(\emptyset)$, we can obtain the binary representation of any coefficient of any monomial in polynomial time in the size of the representation, due to the presence of the advice strings.

Last but not least, for x_i, the representation

$$(e(x_i), A(|(e(x_i), e(x_i))|), \ldots, A(0))$$

is polynomial in i, since $\|x_i\|_e$ is polynomial in i. Thus, we have a representation of the polynomials that satisfies all three conditions. However, we need an encoding and

therefore cannot allow multiple representations of the same value but we must choose exactly one. This is easily fixed though, by choosing that representation of p that has the shortest ΣBF formula α, breaking ties using the lexicographical ordering. We can verify that this only makes representations smaller and use this encoding as e_∞. □

Theorem 116. *Let $\mathcal{R} \neq \mathbb{T}$ be a commutative semiring. If there is an encoding function e for $\mathcal{R}[(x_i)_\infty]$ s.t.*

1) *$\|[\![\alpha]\!]_{\mathcal{R}[(x_i)_\infty]}(\emptyset)\|_e$ is polynomial in the size of α,*
2) *we can extract the encoding $e(r') \in e(R)$ of the coefficient of $x_{i_1}^{j_1} ... x_{i_n}^{j_n}$ from $e(r)$ in time polynomial in $\|r\|_e$, and*
3) *$\|x_i\|_e$ is polynomial in i,*

then either NP $\subseteq$ P/poly or $\text{MOD}_p\text{P} \subseteq$ P/poly for some $p \in \mathbb{N}$.

Theorem 117. *Let $\mathcal{R} \neq \mathbb{T}$ be a commutative semiring. If there is an encoding function e for $\mathcal{R}[x]$ s.t.*

1) *$\|[\![\alpha]\!]_{\mathcal{R}[x]}(\emptyset)\|_e$ is polynomial in the size of α,*
2) *we can extract the encoding $e(r') \in e(R)$ of the coefficient of x^i from $e(r)$ in time polynomial in $\|r\|_e$, and*
3) *$\|x^i\|_e$ is polynomial in $\log_2(i)$,*

then either NP $\subseteq$ P/poly or $\text{MOD}_p\text{P} \subseteq$ P/poly for some $p \in \mathbb{N}$.

Proof of Theorem 116 and 117. As in the proof of Theorem 105, we use that for any non-trivial commutative semiring $e(\mathcal{R})$ it holds that either

1. $k \cdot e(e_\otimes) = e(e_\oplus)$ implies $k = 0$ or
2. $e(\langle e_\otimes \rangle) \equiv \mathbb{Z}_p$ for some $p \in \mathbb{N}$.

In the first case we can derive NP $\subseteq$ P/poly, in the second $\text{MOD}_p\text{P} \subseteq$ P/poly from the preconditions of the theorems.

To prove this, we can reuse the ΣBFs from the proofs of Theorem 113 and Theorem 114 for Theorem 116 and 117, respectively. The size restrictions are given as before. Apart from that, we only need to use that MOD_p3CNF is MOD_pP-complete.

Then, we can see as before that we can read off the solution of any (MOD_p)3CNF-instance of size at most n as a coefficient of $[\![\alpha_n]\!]_{e(\mathcal{R}[(x_i)_\infty)}(\emptyset)$. Here, we note again, that as in the proof of Theorem 105 we only need to recognize a fixed set of values - which is possible in constant time - to derive the result of the (MOD_p)3CNF-instance from the coefficient. □

Lemma 118. *If* $\text{MOD}_p\text{P} \subseteq \text{P/poly}$ *for* $p \in \mathbb{N}, p > 1$*, then* $\text{NP} \subseteq \text{P/poly}$.

Proof. We make use of the following problem:

Problem:	UNIQUE-SAT
Input:	A propositional formula ϕ with at most one satisfying assignment.
Output:	True, if ϕ is satisfiable, otherwise false.

Note that this is a *promise problem*: the input is guaranteed to have at most one satisfying assignment, if this is not the case any output, including non-termination are allowed. It follows, that for any $p \in \mathbb{N}, p > 1$ we can solve UNIQUE-SAT in MOD_pP by asking whether the number of satisfying assignments of the input is not equivalent to zero modulo p.

Valiant and Vazirani [VV86] showed that $\text{NP} \subseteq \text{RP}^{\text{UNIQUE-SAT}}$. Furthermore, recall that RP is the class of languages L for which there exists a polynomial time non-deterministic Turing machine M such that if $x \in L$ then at least half of the computation paths of $M(x)$ accept and if $x \notin L$ then none of the computation paths accept. Intuitively, the definition of RP is more restrictive than that of BPP because if a computation path of the RP-machine M that recognizes L accepts on x, we *know* that $x \in L$, whereas for a BPP-machine, this means that $x \in L$ is likely but not necessarily the case. Indeed, $\text{RP} \subseteq \text{BPP}$, since if L is in RP due to NTM M, then we can prove it is also in BPP by executing M twice independently on a given input x.

Using these insights, it follows that $\text{NP} \subseteq \text{RP}^{\text{UNIQUE-SAT}} \subseteq \text{BPP}^{\text{UNIQUE-SAT}}$. Since UNIQUE-SAT is in MOD_pP for any $p \in \mathbb{N}, p > 1$, we can replace the UNIQUE-SAT-oracle with a MOD_pP-oracle and obtain $\text{NP} \subseteq \text{BPP}^{\text{MOD}_p\text{P}}$.

It remains to show that $\text{BPP}^{\text{MOD}_p\text{P}} \subseteq \text{P/poly}$ follows from $\text{MOD}_p\text{P} \subseteq \text{P/poly}$. We already know that $\text{BPP} \subseteq \text{P/poly}$ [BG81] with an argument based on machine simulation. If $\text{MOD}_p\text{P} \subseteq \text{P/poly}$ holds, we then obtain $\text{BPP}^{\text{MOD}_p\text{P}} \subseteq \text{P}^{\text{P/poly}}\text{/poly}$. To prove the result, it thus remains to show that $\text{P}^{\text{P/poly}}\text{/poly} \subseteq \text{P/poly}$.

Consider a language L that is in $\text{P}^{\text{P/poly}}\text{/poly}$ and solved by a polynomial time machines M with advice oracle A_M, where M uses a further oracle that is evaluated by a polynomial time machine O that uses and advice oracle A_O. Given that $\text{P}^{\text{P}} = \text{P}$, we can combine M and O into a single polynomial time machine M_C and only need to make sure that we can supply it with the advice from both A_M and A_O. The latter is, however, easily achieved. First note that since M takes polynomial time the queries posed to O are all of polynomial size $p(n)$ in the size n of the input to M. Thus, given input x of size n, we can use the following polynomial size advice for M_C:

$$(A_M(n), A_O(p(n)), A_O(p(n)-1), \ldots, A_O(0)).$$

This is still polynomial in n and we have both the necessary advice for the part of M and of O. □

Possibility results for $\mathcal{R}[(x_i)_k]$
↪ Theorem 119

Theorem 119. *Let $e(\mathcal{R})$ be a commutative semiring that is efficiently encoded. Then SAT($e(\mathcal{R}[(x_i)_k])$) is $\mathrm{FP}_{\|}^{NP(e(\mathcal{R}))}$-complete for metric reductions, if we extend e to $\mathcal{R}[(x_i)_k]$ by representing polynomials as lists of monomials with exponents in unary and coefficients encoded by e.*

Proof. Let α be some ΣBF over $e(\mathcal{R}[(x_i)_k])$. We can bound the number of monomials in $[\![\alpha]\!]_{e(\mathcal{R}[(x_i)_k])}(\emptyset)$ by the monomials that occur in α. Let n be the maximum exponent e_i occurring in the monomials $x_1^{e_1} \dots x_k^{e_k}$ in α that have a nonzero coefficient. Furthermore, let m be the number of monomials in α with nonzero coefficients. Then we know that for all monomials $x_1^{e_1} \dots x_k^{e_k}$ in $[\![\alpha]\!]_{e(\mathcal{R}[(x_i)_k])}(\emptyset)$ with nonzero coefficients it holds that $e_i \leq nm$ since every product can only have m factors, where each factor has at most exponent n. Therefore, the number of monomials with nonzero coefficients in $[\![\alpha]\!]_{e(\mathcal{R}[(x_i)_k])}(\emptyset)$ is bounded by $(nm)^k$. Since k is a constant and n and m are polynomial in the size of the input since the exponents are encoded in unary, we know that there are at most polynomially many monomials in $[\![\alpha]\!]_{e(\mathcal{R}[(x_i)_k])}(\emptyset)$ with nonzero coefficients. Finding each of their coefficients is possible with a call to a $\mathrm{NP}(e(\mathcal{R}))$-oracle that is independent of the other calls. This shows that $\mathrm{SAT}(e(\mathcal{R}[(x_i)_k]))$ is in $\mathrm{FP}_{\|}^{\mathrm{NP}(e(\mathcal{R}))}$.

For $\mathrm{FP}_{\|}^{\mathrm{NP}(e(\mathcal{R}))}$-hardness, it is sufficient to show that we can obtain the answers to polynomially many $\mathrm{NP}(e(\mathcal{R}))$-queries by a metric reduction to solving one $\mathrm{SAT}(e(\mathcal{R}[(x_i)_k]))$-instance. Since $\mathrm{SAT}(e(\mathcal{R}))$ is $\mathrm{NP}(e(\mathcal{R}))$-complete, we may assume that we have polynomially many $\mathrm{SAT}(e(\mathcal{R}))$-instances $\alpha_1, \dots, \alpha_n$ and moreover that each α_i in prefix normal form with the same quantifier prefix $\Sigma v_1 \dots \Sigma v_m$. We then construct the following $\mathrm{SAT}(e(\mathcal{R}[(x_i)_k]))$-instance β:

$$\Sigma v_1 \dots \Sigma v_m \, \alpha_1 * e(x_1) + \dots + \alpha_n * e(x_1^n).$$

The coefficient of x_1^i is then the solution of α_i. As β is constructible in time polynomial in the size of $\alpha_1, \dots, \alpha_n$, we have the desired metic reduction, which opmpletes the proof. □

Possibility results for commutative finitely generated semirings
↪ Theorem 122

Theorem 122. *Let $e(\mathcal{R})$ be an efficiently encoded commutative semiring that is generated by $\{r_1, \dots, r_k\}$. Suppose every $r \in R$ is of the form $r = \bigoplus_{i=1}^{n} a_i \cdot \bigotimes_{j=1}^{k} r_j^{e_{i,j}}$ for some $a_i, e_{i,j} \in \mathbb{N}$ such that*

- $\max\{e_{i,j}, \log_2(a_i), n\}$ *is polynomial in $\|r\|_e$, and*
- *we can obtain $a_i, e_{i,j}$ from $e(r)$ in polynomial time.*

If $e(\mathcal{R})$ *is*

1. *periodic with periodicity* 1*;*
2. *periodic with periodicity* $p \geq 2$ *and offset* 0*;*
3. *periodic with periodicity* $p \geq 2$ *and offset* $o > 0$*;*
4. *not periodic;*

then $SAT(e(\mathcal{R}))$ *is in*

1. $FP_{\|}^{NP}$*;*
2. $FP_{\|}^{\text{MOD}_p P}$*;*
3. $FP_{\|}^{\text{MOD}_p P \cup NP}$*;*
4. $FP_{\|}^{\#P}$*;*

respectively.

Proof. Let α a ΣBF over $e(\mathcal{R})$. We can transform α into a ΣBF β over $\text{bin}(\mathcal{S}[(x_i)_k])$, where the binary encoding of the natural numbers bin is extended to the polynomials by representing polynomials as lists of monomials with exponents in unary and coefficients encoded by bin and $\mathcal{S}$ is

1. $\mathbb{N}_{\leq o}$,
2. $\mathbb{Z}_p$,
3. $\mathbb{Z}_p \times \mathbb{N}_{\leq o}$,
4. $\mathbb{N}$

if $e(\mathcal{R})$ is, respectively,

1. periodic with periodicity 1 and offset o
2. periodic with periodicity $p \geq 2$ and offset 0
3. periodic with periodicity $p \geq 2$ and offset $o > 0$
4. not periodic.

The transformation works by replacing every value $e(r) \in e(R)$ that occurs in α by

$$\mathrm{bin}\left(\sum_{i=1}^{n} a_i \prod_{j=i}^{m_i} x_j^{e_{i,j}}\right)$$

when

$$e(r) = \bigoplus_{i=1}^{n} a_i \bigotimes_{j=i}^{m_i} e(r_j)^{e_{i,j}}.$$

Recall that when $k = 0$, i.e., $e(\mathcal{R})$ is already generated by the empty set, these equations collapse to

$$\mathrm{bin}\left(\sum_{i=1}^{n} a_i \cdot 1\right)$$

and

$$e(r) = \bigoplus_{i=1}^{n} a_i \cdot e_{\otimes}$$

since we take the neutral element of multiplication as the value of an empty product.

During the above transformation, we only need to check whether a_i is in $\mathcal{S}$ and replace a_i by an equivalent value if not. Namely, if $\mathcal{S} = \mathbb{N}_{\leq o}$, we use the minimum of a_i and o, if $\mathcal{S} = \mathbb{Z}_p$, we use the unique integer n in $\{0, \ldots, p-1\}$ such that $a_i \equiv n \mod p$, if $\mathcal{S} = \mathbb{Z}_p \times \mathbb{N}_{\leq o}$, we use (n, m), where n is the unique integer in $\{0, \ldots, p-1\}$ such that $a_i \equiv n \mod p$ and m is the minimum of a_i and o, otherwise, $\mathcal{S} = \mathbb{N}$ in which case we know that $a_i \in \mathbb{N}$. We can obtain the above expression in polynomial time. Hence, the size of β is also polynomial in the input.

Theorem 119 tells us, that $\mathrm{SAT}(\mathrm{bin}(\mathcal{S}[(x_i)_k]))$ satisfies the desired complexity assertion. Furthermore, given $[\![\beta]\!]_{\mathrm{bin}(\mathcal{S}[(x_i)_k])}(\emptyset)$ we can compute $[\![\alpha]\!]_{e(\mathcal{R})}(\emptyset)$ in polynomial time. This can be seen as follows. Let

$$[\![\beta]\!]_{\mathcal{S}[(x_i)_k]}(\emptyset) = \sum_{i_1,\ldots,i_k=1}^{n} a_{\bar{i}} \prod_{j=1}^{k} x_j^{e_{\bar{i},j}}.$$

Then we know that

$$[\![\alpha]\!]_{\mathcal{R}}(\emptyset) = \bigoplus_{i_1,\ldots,i_k=1}^{n} \bigoplus_{v=1}^{a_{\bar{i}}} \bigotimes_{j=1}^{k} r_j^{e_{\bar{i},j}}.$$

Since the size of the representation of $[\![\beta]\!]_{\mathrm{bin}(\mathcal{S}[(x_i)_k])}(\emptyset)$ is polynomial, also the number of monomials in $[\![\beta]\!]_{\mathrm{bin}(\mathcal{S}[(x_i)_k])}(\emptyset)$ is polynomial. Therefore, we can calculate the values of all the "monomials" $e(\bigotimes_{j=1}^{k} r_j^{e_{\bar{i},j}})$ in polynomial time.

To compute

$$e(\bigoplus_{v=1}^{a_{\bar{i}}} \bigotimes_{j=1}^{k} r_j^{e_{\bar{i},j}})$$

first consider the binary representation $c_{\lceil \log_2(a_{\bar{i}}) \rceil} \ldots c_0 \in \{0,1\}^*$ of $a_{\bar{i}}$. We know that

$$e(\bigoplus_{v=1}^{a_{\bar{i}}} \bigotimes_{j=1}^{k} r_j^{e_{\bar{i},j}}) = \bigoplus_{t=0}^{\lceil \log_2(a_{\bar{i}}) \rceil} c_t \cdot e(\bigoplus_{v=1}^{2^t} \bigotimes_{j=1}^{k} r_j^{e_{\bar{i},j}}). \tag{B.2}$$

Thus, we can proceed by first iteratively computing

$$e(\bigoplus_{v=1}^{2^t}\bigotimes_{j=1}^{k} r_j^{e_{\bar{i},j}}) = e(\bigoplus_{v=1}^{2^{t-1}}\bigotimes_{j=1}^{k} r_j^{e_{\bar{i},j}}) \oplus e(\bigoplus_{v=1}^{2^{t-1}}\bigotimes_{j=1}^{k} r_j^{e_{\bar{i},j}})$$

for $t = 1, \ldots, \lceil \log_2(a_{\bar{i}}) \rceil$. This is possible in $\lceil \log_2(a_{\bar{i}}) \rceil$ steps, where each of them takes polynomial time and leads to a result that is polynomial in size, since $e(\mathcal{R})$ is efficiently encoded.

After having computed these values, we can plug them into Equation (B.2) and evaluate it. This is again possible in polynomial time, and leads to a result that is polynomial in size, since $e(\mathcal{R})$ is efficiently encoded. For the same reason, summing up all the values $e(\bigoplus_{v=1}^{a_{\bar{i}}}\bigotimes_{j=1}^{k} r_j^{e_{\bar{i},j}})$ over $\bar{i}$ is also possible in polynomial time and leads to a polynomial-size result.

From metric-reducibility to $\mathrm{SAT}(\mathcal{S}[(x_i)_k])$ and Theorem 119, it follows that $\mathrm{SAT}(e(\mathcal{R}))$ is in $\mathrm{FP}_{\parallel}^{\mathrm{FP}^{\mathcal{C}}}$. The latter class is equal to $\mathrm{FP}_{\parallel}^{\mathcal{C}}$, which complete the proof. □

B.3.4 Derived results
↪ Theorem 123

Theorem 123. *Let $\mathbb{S} = \mathbb{N}, \mathbb{Z}, \mathbb{Q}$. For $\mathbb{S}^n$, $n \in \mathbb{N}$, the semiring $\mathbb{S}$ over multiple dimensions, we have that SAT(bin($\mathbb{S}$)^n) is $\mathrm{FP}_{\parallel}^{\#P}$-complete with respect to metric reductions, where bin($\mathbb{Q}$) represents $r \in \mathbb{Q}$ as pair (bin(p), bin(q)) such that $p/q = r$, $p \in \mathbb{Z}$, $q \in \mathbb{N}$ and the greatest common divisor of $|p|$ and q is 1.*

Proof. For $\mathbb{S} = \mathbb{N}$ and $\mathbb{S} = \mathbb{Z}$ we see that $\mathbb{S}^n$ is finitely generated by $\{e_1, \ldots, e_n\}$ and $\{e_1, \ldots, e_n, (-1, \ldots, -1)\}$, respectively, where e_i is the vector whose j^{th} entry is 0 if $j \neq i$ and 1 if $i = j$. Furthermore, every number $m = (m_1, \ldots, m_n) \in \mathbb{S}$ can be represented as a sum

$$m = \sum_{i=1}^{n} \sum_{j=1}^{|m_i|} \begin{cases} e_i & \text{if } m_i \geq 0, \\ (-1, \ldots, -1)e_i & \text{if } m_i < 0. \end{cases}$$

such that $\log_2(m_i)$ is polynomial in $\|m\|_{\mathrm{bin}}$. Therefore, the preconditions of Theorem 122 are satisfied and we obtain that $\mathrm{SAT}(\mathrm{bin}(\mathbb{S})^n)$ is in $\mathrm{FP}_{\parallel}^{\#\mathrm{P}}$.

For $\mathbb{S} = \mathbb{Q}$, this strategy does not apply completely analogously since $\mathbb{Q}$ is not finitely generated. However, we can apply a very similar strategy as follows.

Consider a $\mathrm{SAT}(\mathrm{bin}(\mathbb{Q})^n)$-instance α. Let $\mathrm{bin}(p_1/q_1), \ldots, \mathrm{bin}(p_m/q_m) \in \mathrm{bin}(\mathbb{Q})$ be the entries of the weight vectors in α. Furthermore, let l be the least common multiple of $q_1, \ldots, q_m$. Then we can replace these values by polynomials from $\mathrm{bin}(\mathbb{N}[(x_i)_2])$ by using

$$\mathrm{poly}(p_i, q_i) = \begin{cases} \mathrm{bin}(p_i l / q_i x_2) & \text{if } p_i \geq 0, \\ \mathrm{bin}(|p_i| l / q_i x_1 x_2) & \text{if } p_i < 0. \end{cases}$$

for $\mathrm{bin}(p_i/q_i)$. Note that when we replace x_1 by -1 and x_2 by $1/l$, we get the original value again. The same still holds after addition and multiplication, that is, $f : \mathrm{bin}(\mathbb{N}[(x_i)_2]) \to$

$\mathrm{bin}(\mathbb{Q}), p(x_1, x_2) \mapsto p(-1, l)$ is a homomorphism. In fact, when we restrict the codomain of f to $\mathrm{bin}(\langle 1/l, -1\rangle$, it is even an epimorphism, which is moreover computable in polynomial time.

Thus, we can metrically reduce $\mathrm{SAT}(\mathrm{bin}(\mathbb{Q})^n)$ to $\mathrm{SAT}(\mathrm{bin}(\mathbb{N}[(x_i)_2])^n)$. However, the semiring $\mathrm{bin}(\mathbb{N}[(x_i)_2])^n$ is finitely generated, by $\{e_i, e_i x_1, e_i x_2 \mid i = 1, \ldots, n\}$ and satisfies the preconditions of Theorem 122. Thus, $\mathrm{SAT}(\mathrm{bin}(\mathbb{N}[(x_i)_2)^n)$ and consequently also $\mathrm{SAT}(\mathrm{bin}(\mathbb{Q})^n)$ is in $\mathrm{FP}_{\|}^{\#\mathrm{P}}$.

Regarding hardness, note that already $\mathrm{SAT}(\mathrm{bin}(\mathbb{N}))$ is $\mathrm{FP}_{\|}^{\#\mathrm{P}}$-hard with respect to metric reductions: Given n $\mathrm{SAT}(\mathrm{bin}(\mathbb{N}))$-instances $I_1, \ldots, I_n$ we can compute the value of the $\mathrm{SAT}(\mathrm{bin}(\mathbb{N}))$-instance $\alpha_n = I_1 * \mathrm{bin}(2^k) + \ldots + I_n * \mathrm{bin}(2^{k*n})$, where k is such that $2^k > [\![I_i]\!]_{\mathrm{bin}(\mathbb{N})}(\emptyset)$ for all i. Such a k can be found and is polynomial in the size of the instances. Due to the choice of k we can understand the result of α_n as a list of the results of I_i in binary representation. From the $\mathrm{FP}_{\|}^{\#\mathrm{P}}$-hardness it immediately follows, that $\mathrm{SAT}(\mathrm{bin}(\mathbb{S})^n)$ is even $\mathrm{FP}_{\|}^{\#\mathrm{P}}$-complete with respect to metric reductions for $\mathbb{S} = \mathbb{N}, \mathbb{Z}, \mathbb{Q}$. □

APPENDIX C

Full Proofs: Efficient Algebraic Answer Set Counting

C.1 Proofs Regarding Clark's Completion ↪ Theorem 149

Theorem 149. *Given a normal answer set program Π and a tree decomposition $\mathcal{T} = (T, \chi)$ of $INC(\Pi)$ with width k and root t_r, the CNF $IClark(\Pi, \mathcal{T}, t_r)$ can be constructed in time linear in $|\Pi| + |T|$ and satisfies that*

(i) *every model of $Clark_{Prop}(\Pi)$ can be uniquely extended to a model of $IClark(\Pi, \mathcal{T}, t_r)$,*

(ii) *the size of $IClark(\Pi, \mathcal{T})$ is in $\mathcal{O}(k|\Pi|)$, and*

(iii) *the treewidth of $PRIM(IClark(\Pi, \mathcal{T}, t_r))$ is at most $3(k+1)$.*

Proof. We can assume w.l.o.g. that every node $t \in T$ has at most 2 children and that T has at most $|\Pi|$ nodes. Otherwise, we transform the tree decomposition into an equivalent one, where this is the case. This is possible in linear time in $|T|$ [Klo94].

In order to obtain a CNF we express every equivalence $aux \leftrightarrow \bigvee_{x \in X}$ in Definition 148 by the clauses $\neg aux \vee \bigvee_{x \in X} x$ and $aux \vee \neg x$ for $x \in X$. Similarly, for $aux \leftrightarrow \bigwedge_{x \in X}$ we use the clauses $aux \vee \bigvee_{x \in X} \neg x$ and $\neg aux \vee x$ for $x \in X$.

For 1. first recall that the Tseitin transformation has the desired property. Then observe that the given translation partially applies the Tseitin transformation on the formula

$$a \leftrightarrow \bigvee_{r \in \Pi, head(r) = a} \bigwedge_{l \in body(r)} l$$

for every $a \in \mathcal{A}(\Pi)$. Since Clark's Completion is the conjunction of the above formula for every $a \in \mathcal{A}(\Pi)$, this proves the desired claim.

2. follows immediately from Definition 148. We introduce one auxiliary atom $forced_r$ for every rule in Π and one auxiliary atom $upto_t^x$ for every combination x, t such that $x \in \chi(t)$. Last but not least, observe that we have at most six equivalences for each pair x, t such that $x \in \chi(t)$. Furthermore, each of these equivalences involves at most $k+2$ variables, since every node has at most two children. Thus, the clauses corresponding to the equivalences have at most $k+2$ atoms and there are at most $6 * (k+2)^2$ many of them for each node $t \in T$. Since $|T| \leq |\Pi|$, we are done.

In order to prove 3. we construct a tree decomposition $\mathcal{T}' = (T', \chi')$ of PRIM(C) from the given tree decomposition of INC(Π). Namely, we assume that $\chi(t) = \{x_1^t, \ldots, x_{|\chi(t)|}^t\}$ and use $T' = (V', E')$, where

$$\begin{aligned} V' &= \{(t,i) \mid t \in T, i = 1, \ldots, |\chi(t)|\} \\ E' &= \{((t,i+1),(t,i)) \mid t \in V, i = 1, \ldots, |\chi(t)| - 1\} \\ &\quad \cup \{((t,1),(t',|\chi(t')|)) \mid (t,t') \in E\} \end{aligned}$$

and

$$\begin{aligned} \chi'(t,i) =& \chi(t) \cap \mathcal{A}(\Pi) \cup \{forced_r \mid r \in \chi(t)\} \\ &\cup \{upto_{t'}^{x_j^t} \mid t' \in children(t), x_j^t \in \chi(t'), j \geq i\} \cup \{upto_t^{x_j^t} \mid j \leq i\}. \end{aligned}$$

Then it holds that

$$\begin{aligned} |\chi'(t,i)| &= |\chi(t) \cap \mathcal{A}(\Pi)| + |\{forced_r \mid r \in \chi(t)\}| \\ &\quad + |\{upto_{t'}^{x_j^t} \mid t' \in children(t), x_j^t \in \chi(t'), j \geq i\}| + |\{upto_t^{x_j^t} \mid j \leq i\}| \\ &\leq |\chi(t)| + 2(k+1-i+1) + i \\ &\leq 3(k+1) + 1. \end{aligned}$$

Therefore, the width of $\mathcal{T}'$ is less or equal to $3(k+1)$. Furthermore, $\mathcal{T}'$ is a tree decomposition of PRIM(C), since for each equivalence in Definition 148 there is a bag that contains all the variables that occur in it. □

C.2 Proofs Regarding Cycle Breaking ↪ Lemmas 151, 157, 159, Theorems 153, 155, 154, 162

Lemma 151 (Faithfulness Implies Query Invariance)**.** *Let $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$ be a measure and let $\mathcal{C}(.)$ be a faithful cycle breaking for Π. Then for $\mu' = \langle \mathcal{C}(\Pi), \alpha, \mathcal{R} \rangle$ it holds that $\mu(a) = \mu'(a)$ for every $a \in \mathcal{A}(\Pi)$.*

Proof. Recall that a cycle breaking $\mathcal{C}(.)$ is faithful (for Π), if:

- $|\mathcal{AS}(\Pi)| = |\mathcal{AS}(\mathcal{C}(\Pi))|$ and,
- $\mathcal{AS}(\Pi) = \{\mathcal{I} \cap \mathcal{A}(\Pi) \mid \mathcal{I} \in \mathcal{AS}(\mathcal{C}(\Pi))\}$.

Let $\mathcal{C}(.)$ be a faithful cycle breaking (for Π). Then we can define a bijective map f between $\mathcal{AS}(\Pi)$ and $\mathcal{AS}(\mathcal{C}(\Pi))$, such that for each interpretation $\mathcal{I} \subseteq \mathcal{A}(\Pi)$ the answer set $f(\mathcal{I})$ of $\mathcal{C}(\Pi)$ satisfies $\mathcal{I} = \mathcal{A}(\Pi) \cap f(\mathcal{I})$.

Now let $a \in \mathcal{A}(\Pi)$. We know that $\mu'(a)$ is the sum of $[\![\alpha]\!]_{\mathcal{A}}(f(\mathcal{I}))$ over all answer sets $f(\mathcal{I})$ of Π, since f is bijective. Since α contains only variables in $\mathcal{A}(\Pi)$ it holds that $[\![\alpha]\!]_{\mathcal{A}}(f(\mathcal{I})) = [\![\alpha]\!]_{\mathcal{A}}(f(\mathcal{I}) \cap \mathcal{A}(\Pi))$. However, $\mathcal{I} = f(\mathcal{I}) \cap \mathcal{A}(\Pi) = \mathcal{I}$ is guaranteed to be an answer set of Π. This implies that $\mu'(a) = \mu(a)$. □

Theorem 153. *There is a family of programs $(\Pi_n)_{n\in\mathbb{N}}$ such that*

1. *DEP(Π_n) has exactly one simple cycle,*
2. *the treewidth of Π_n is bounded by a constant (independent of n),*
3. *the number of atoms and rules of Π_n is linear in n, and*
4. *the treewidth of MJ(Π_n) grows linearly with n.*

Proof. For this, we define

$$\begin{aligned}\Pi_n = &\{\{v(i)\} \leftarrow \mid i = 1, \ldots, n\} \\ &\cup \{\{e(i,j)\} \leftarrow \mid i, j = 1, \ldots, n, i+1 \equiv j \mod n\} \\ &\cup \{in(i) \leftarrow v(i) \mid i = 1, \ldots, n\} \\ &\cup \{in(i) \leftarrow e(i,j), in(j) \mid i, j = 1, \ldots, n, i+1 \equiv j \mod n\}.\end{aligned}$$

Intuitively, Π_n takes the directed graph over n vertices with arcs

$$(1,2), (2,3), \ldots, (n-1,n), (n,1),$$

thus inducing exactly one cycle $1, 2, \ldots, n, 1$. Then it guesses a random subset of it. All vertices are kept such that $v(i)$ holds or the edge of the predecessor is present $e(i,j)$ and the vertex j was kept.

For all n it holds that DEP(Π_n) has exactly one simple cycle, namely

$$in(1), in(2), \ldots, in(n), in(1).$$

In order to prove that the treewidth of PRIM(Π_n) is bounded by a constant, we define a tree decomposition (T_n, χ_n) as follows:

$$T_n = (V_n, E_n),$$

$$
\begin{aligned}
V_n &= \{t_i \mid i = 1, \ldots, n\},\\
E_n &= \{(t_i, t_{i+1}) \mid i = 1, \ldots, n-1\},\\
\chi_n(t_i) &= \begin{cases}
\{v(i), nv(i), in(i), in(i+1),\\
\quad e(i, i+1), ne(i, i+1), in(1)\} & \text{if i = 1, \ldots, n - 1,}\\
\{v(n), nv(n), in(n), in(1), e(n, 1), ne(n, 1)\} & \text{if } i = n.
\end{cases}
\end{aligned}
$$

First note that (T_n, χ_n) is a tree decomposition of PRIM(Π), since every rule is contained in a bag completely and every subgraph of T_n induced by taking only those vertices t_i whose bag contains an atom a is connected. Furthermore, the width of (T_n, E_n) is 6 and therefore bounded by a constant independent of n.

It remains to show that the treewidth of PRIM(MJ(Π_n)) grows linearly with n. For this, we first construct the program MJ(Π_n):

$$
\begin{aligned}
&\{\{\{v(i)\} \leftarrow \mid i = 1, \ldots, n\}\\
&\cup \{\{e(i, j)\} \mid i, j = 1, \ldots, n, i + 1 \equiv j \mod n\}\\
&\cup \{in(i)_F \leftarrow v(i) \mid i = 1, \ldots, n, F \in \mathrm{ch}_i\}\\
&\cup \{in(i)_F \leftarrow e(i, j), in(j)_{F \cup \{in(i)\}} \mid i, j = 1, \ldots, n, i + 1 \equiv j \bmod n, F \in \mathrm{ch}_i\},
\end{aligned}
$$

where ch_i denotes the set of chains of atoms that can be used to derive in(i), given by

$$
\begin{aligned}
\{\emptyset, &\{in(i-1)\}, \{in(i-1), in(i-2)\}, \ldots,\\
&\{in(i-1), \ldots, in(1), in(n), \ldots, in(i+1)\}\}.
\end{aligned}
$$

Then the primal graph of MJ(Π_n) contains at least the following edges:

1. $\{in(i)_F, in(j)_{F \cup \{in(i)\}}\}$ for $i, j = 1, \ldots, n$ and $F \in \mathrm{ch}_i$ such that $i + 1 \equiv j \mod n$ and $in(j) \notin F$, and
2. $\{in(i)_F, e(i, j)\}$ for $i, j = 1, \ldots, n$ and $F \in \mathrm{ch}_i$ such that $i + 1 \equiv j \mod n$ and $in(j) \notin F$.

In order to show lower bounds on the width of every tree decomposition of the primal graph, we need the following two facts:

Min Degree Lower Bound [KBvH01] The treewidth of a graph G is at least its smallest node degree, i.e.,

$$\min_{v \in V(G)} |\{v' \mid \{v, v'\} \in E(G)\}|.$$

Minor Monotonicity [Bod98] Given a graph G, the treewidth of any graph minor G' of G, i.e. a graph obtained from G by deleting vertices or edges or by contracting edges, is at most as high as the treewidth of G.

This means that it is sufficient to show that there is a graph minor of the primal graph of Π_n such that the minimum degree of a vertex in it is in $\Omega(n)$.

We proceed as follows. First, we contract all the edges of the first form. Since, we have for each $i = 1, \ldots, n$ an edge sequence

$$\begin{aligned}
&\{in(i)_\emptyset, in(i+1)_{\{in(i)\}}\},\\
&\{in(i+1)_{\{in(i)\}}, in(i+2)_{\{in(i),in(i+1)\}}\},\\
&\ldots,\\
&\{in(i-2)_F, in(i-1)_{F\cup\{in(i-2)\}}\}
\end{aligned}$$

that we contract, the vertex $res(i)$ resulting from contracting it has neighbors

$$\{e(i', j') \mid i', j' = 1, \ldots, n, i' \neq i, i' + 1 \equiv j' \mod n\}$$

due to the second kind of edges.

Similarly, $e(i, i+1)$ for $i = 1, \ldots, n-1$ has neighbors $\{res(i') \mid i' \neq i+1\}$. Thus, if we remove every other vertex that is not of the form $res(i)$ or $e(i, j)$, then every vertex in the remaining graph minor has at least $n-1$ neighbors. From the min-degree lower bound it follows that the primal graph of $\mathrm{MJ}(\Pi_n)$ has treewidth at least $n-1$. □

Theorem 154. *Let Π be a normal answer set program of treewidth k such that the largest strongly connected component of DEP(Π) has size s. Then the treewidth of JN(Π) is in $\mathcal{O}(k^2 + k \log_2(s))$.*

Proof. Let (T, χ) be an optimal tree decomposition of $\mathrm{PRIM}(\Pi)$. We can modify it to be a tree decomposition of $\mathrm{PRIM}(\mathrm{JN}(\Pi))$, resulting in (T', χ'), where

$$\begin{aligned}
T' =& T,\\
\chi'(t) =& \chi(t) \cup \{\mathbf{next}(a), \mathbf{just}(a) \mid a \in \chi(t)\} \cup \bigcup_{a\in\chi(t)} \mathrm{bin}(a)\\
&\cup \{\mathbf{lt}(b, a), \mathbf{succ}(b, a) \mid a, b \in \chi(t), \mathbf{lt}(b, a) \in \mathcal{A}(\mathrm{JN}(\Pi))\}.
\end{aligned}$$

Here, $\mathrm{bin}(a)$ refers to the set of variables that are used to encode the binary counter for $a \in \mathcal{A}(\Pi)$. For each $a \in \mathcal{A}(\Pi)$ there are at most $\lceil \log_2(s) \rceil$ such variables.

Observe that the width upper bound holds, since

$$\begin{aligned}
|\chi'(t)| &= |\chi(t)| + |\{\mathbf{next}(a), \mathbf{just}(a) \mid a \in \chi(t)\}| + \sum_{a\in\chi(t)} |\mathrm{bin}(a)|\\
&\quad + |\{\mathbf{lt}(b, a), \mathbf{succ}(b, a) \mid a, b \in \chi(t), \mathbf{lt}(b, a) \in \mathcal{A}(\mathrm{JN}(\Pi))\}|\\
&\leq k + k + k\lceil \log_2(s) \rceil + k^2
\end{aligned}$$

The last expression is in $\mathcal{O}(k^2 + k \log_2(s))$, as claimed.

It remains to show that (T', χ') is a tree decomposition of PRIM(JN(Π)). This follows form the fact that (T, χ) is a tree decomposition of PRIM(Π). Therefore, for every rule $r \in \Pi$ there exists a bag $t \in T$ such that $head(r) \cup body(r) \subseteq \chi(t)$. In JN(Π) we only add rules that are based on an original rule $r \in \Pi$, meaning that they only use atoms from r or auxiliary atoms that are related to atoms in r. I.e., when an additional rule uses any of $\mathbf{next}(a), \mathbf{just}(a), \mathbf{lt}(b, a)$ or $\mathbf{succ}(b, a)$ then both a and b must occur in r. Since we defined $\chi'(t)$ in such a way that it includes all auxiliary atoms that are related to the atoms in $\chi(t)$ this means all additional rules derived from a rule r are completely contained in $\chi'(t)$, when r is completely contained in $\chi(t)$. □

Lemma 157. *Let* Π *be an answer set program and* $s \in \mathcal{A}(\Pi)^*$ *be an unfolding sequence. Then* $UF(DEP(\Pi), s) = DEP(T_{\mathcal{P}}\text{-}Unfold(\Pi, s))$ *(when identifying* a *with* $a^{cnt(a,s)}$*).*

Proof. We proof this lemma by induction on the length of s. If $|s| = 0$, then the program $T_{\mathcal{P}}$-Unfold(Π, s) contains only constraints and thus DEP($T_{\mathcal{P}}$-Unfold(Π, s)) is the empty digraph. On the other hand, UF(DEP(Π), s) is also equal to the empty digraph.

Next, we prove the induction step. Assume the lemma holds for all s of length up to n. Now let $s = s's_{n+1}$ an unfolding sequence of length $n+1$. Then $T_{\mathcal{P}}$-Unfold(Π, s) is equal to the union of $T_{\mathcal{P}}$-Unfold(Π, s') and the rules that were added in the $n+1$-th iteration (modulo renaming of $s_{n+1}^{cnt(s_{n+1})}$ to s_{n+1}). Thus we can apply the induction hypothesis and obtain that

$$\text{DEP}(T_{\mathcal{P}}\text{-Unfold}(\Pi, s')) = \text{UF}(\text{DEP}(\Pi), s').$$

We now consider the additional rules that were added in the $n+1$-th iteration. Then

$$\begin{aligned} V(\text{DEP}(T_{\mathcal{P}}\text{-Unfold}(\Pi, s))) &= V(\text{DEP}(T_{\mathcal{P}}\text{-Unfold}(\Pi, s'))) \cup \{s_{n+1}^{cnt(s_{n+1})}\} \\ &= V(\text{UF}(\text{DEP}(\Pi), s')) \cup \{s_{n+1}^{cnt(s_{n+1})}\} \\ &= V(\text{UF}(\text{DEP}(\Pi), s)). \end{aligned}$$

Furthermore, the edges that are added to the dependency graph after the $n+1$-th iteration are of the form (b, s_{n+1}) for $(b, s_{n+1}) \in E(\text{DEP}(\Pi))$, since we only add rules that derive s_{n+1} and because we only add rule copies (see line 10). On the other hand, if (1) $(b, s_{n+1}) \in E(\text{DEP}(\Pi))$ and (2) b occurred in s', then we add an edge (b, s_{n+1}) as (1) implies that there is a rule that uses b to derive s_{n+1} and (2) implies that a copy of the rule is added that has b in the body (as $last(b) = b$). This exactly correspond to the edges that we define to be in $E(\text{UF}(\text{DEP}(\Pi), s))$ for $k = n + 1$. □

Theorem 155. *For any factorized measure* $\mu = \langle \Pi, \alpha, \mathcal{R} \rangle$*, we can construct in polynomial time in the size of* Π *given access to an NP-oracle a factorized measure* $\mu' = \langle \Pi', \alpha, \mathcal{R} \rangle$ *with an acyclic program* Π′ *such that*

(i) *for all* $a \in \mathcal{A}(\Pi)$ *it holds that* $\mu(a) = \mu'(a)$,

(ii) *the treewidth of* Π' *is at most* $k \cdot cbs(DEP(\Pi))$*, where* k *is the treewidth of* Π*, and*

(iii) *the size of* Π' *is at most* $cbs(DEP(\Pi)) \cdot |\Pi|$.

For the proof we use the following auxiliary lemma.

Lemma 159. *Let* Π *be an answer set program with treewidth* k *and* $s \in \mathcal{A}(\Pi)^*$ *be an unfolding sequence. If every variable* $a \in \mathcal{A}(\Pi)$ *occurs at most* m *times in* s*, then the treewidth of* $T_{\mathcal{P}}$-UNFOLD(Π, s) *is less or equal to* $k \cdot m$.

Proof. We know that for each variable $a \in \mathcal{A}(\Pi)$ we introduce at most $m-1$ copies $a_1, \ldots, a_{m-1}$ during unfolding. Now, let (T, χ) be an optimal tree decomposition for Π. We take (T, χ'), where

$$\chi'(t) = \chi(t) \cup \{a_j \mid 1 \leq j \leq m-1, a \in \chi(t)\}.$$

Recall line 11 of Algorithm 2, which is the only line where rules are added to the output program Π'. We see that for each such rule head $\leftarrow B^+_{new}, B^-(r)$ added at the iteration where we consider s_i, it holds that *head* is either a copy of s_i or s_i itself. The same holds for the atoms in B^+_{new}. For $B^-(r)$ we even only have the original atoms.

The first condition for (T, χ') to be a tree decomposition is easily seen to be true. We know that (T, χ) is a tree decomposition for PRIM(Π), therefore, every atom $a \in \mathcal{A}(\Pi)$ is in $\chi(t_a)$. Accordingly, since $\mathcal{A}(\text{T}_{\mathcal{P}}\text{-UNFOLD}(\Pi, s))$ only contains the original atoms a or copies a^i we know that $a, a^i \in \chi(t_a)$.

Next we must check that for every edge $\{x, y\}$ in the primal graph of T$_{\mathcal{P}}$-UNFOLD(Π, s) there is a node t such that $x, y \in \chi'(t)$. Again, we know that for every edge $\{x, y\} \in E(\Pi)$ there is a node $t_{x,y}$ such that $x, y \in \chi(t_{x,y})$. Furthermore, we know that there is an edge between x and y if they both occur in the same rule. Since the rules in T$_{\mathcal{P}}$-UNFOLD(Π, s) are all copies of rules with copies of atoms, we know that if $\{x, y\} \in E(\text{PRIM}(\text{T}_{\mathcal{P}}\text{-UNFOLD}(\Pi, s)))$, where the original atom of x and y are a and b respectively, then $\{a, b\} \in E(\text{PRIM}(\Pi))$. It follows that $x, y \in \chi'(t_{a,b})$.

Last but not least, since the subgraphs induced by removing all bags that do not contain a variable a are trees in the original tree decomposition, this also holds for the new tree decomposition. Thus, (T, χ') is a tree decomposition of T$_{\mathcal{P}}$-UNFOLD(Π, s).

Further, it is easy to see that $|\chi'(t)| \leq |\chi(t)| \cdot m \leq k \cdot m$. □

Theorem 162. *The problem of checking whether* $cbs(G) \leq k$ *given a digraph* G *and* $k \in \mathbb{N}$ *in the input is NP-complete.*

Proof. NP-membership is easy to see by a guess and check algorithm.

For NP-hardness, we use a reduction from SAT, i.e., checking whether a CNF $\mathcal{C}$ has a satisfying assignment. We assume w.l.o.g. that $\mathcal{C}$ does not have any empty clauses. We

construct a digraph G with two vertices v_x and $v_{\neg x}$ for each variable $x \in Vars(\mathcal{C})$ and its negation $\neg x$ and one vertex v_C for each clause C. Then we add arcs to include a cycle between v_x and $v_{\neg x}$ for every variable x, in addition to one cycle $v_{l_1} \dots v_{l_n} v_C$ for each clause $C = l_1 \vee \dots v_n \in \mathcal{C}$.

Claim I: Let $N = |Vars(\mathcal{C})|$. There exists a subset $S \subseteq V(G)$ such that $G \setminus S$ is acyclic of size N iff the original CNF was satisfiable.

This can be seen as follows: Every subset $S \subseteq V(G)$ such that $G \setminus S$ is acyclic must have a size of at least N, since for each variable we need to at least include x or $\not{x}$ in S. Since we additionally have a cycle $v_{l_1} \dots v_{l_n} v_C$ for each clause $C = l_1 \vee \dots v_n \in \mathcal{C}$ every satisfying assignment $\mathcal{I}$ to $\mathcal{C}$ corresponds to a set

$$S_{\mathcal{I}} = \{v_l \mid \mathcal{I} \models l, l = x, \neg x, x \in Vars(\mathcal{C})\}$$

that satisfies $G \setminus S_{\mathcal{I}}$ is acyclic. And additionally for every S of size N (implying that S only contains vertices of the form v_x, $v_{\neg x}$) such that $G \setminus S$ is acyclic, it holds that

$$\mathcal{I}_S = \{x \in Vars(\mathcal{C}) \mid v_x \in S\}$$

is a satisfying assignment to $\mathcal{C}$. This proves the claim.

However, cbs(G) does not ask for the minimum size of a backdoor S such that $G \setminus S$ is acyclic. We can however force every minimum solution for cbs(G) to be of that form by performing an additional modification on G. Namely, we add vertices $extra_1, \dots, extra_N$ and arcs $(v_x, extra_i)$, $(extra_i, v_{\neg x})$ for every $i \in \{1, \dots, N\}$ and $x \in Vars(\mathcal{C})$. An example is shown in Figure C.1. Observe that **Claim I** still holds for the modified graph, since every added cycle uses both v_x and $v_{\neg x}$, thus, removing either of them suffices.

Claim II: Let $G_{\mathcal{C}}$ denote constructed graph. cbs$(G_{\mathcal{C}}) \leq N + 1$ iff $\mathcal{C}$ is satisfiable.

By proving this claim the reduction is completed. We already know that if $\mathcal{C}$ is satisfiable, then some set S exists such that $G_{\mathcal{C}} \setminus S$ is acyclic and thus by case (i) and (iv) of the definition of cbs(.) we have cbs$(G_{\mathcal{C}}) = N+1$, as desired. Assume now that cbs$(G_{\mathcal{C}}) \leq N+1$. As $\mathcal{C}$ contains no empty clauses, $G_{\mathcal{C}}$ is cyclic, strongly connected, and not a polytree (due to the extra vertices and arcs). Thus, we are in case (iv) of the definition of cbs(.) meaning we compute it by choosing $S \subseteq V(G_{\mathcal{C}})$ and multiplying cbs$(G_{\mathcal{C}} \setminus S)$ by $(|S| + 1)$; hence $|S| \leq N$. Consider now the following three observations:

1. for $\{v_x, v_{\neg x}\}$ and $\{v_y, v_{\neg y}\}$ to lie in different SCCs of $G_{\mathcal{C}} \setminus S$ without using vertices of the from $\{v_x, v_{\neg x}, v_y, v_{\neg y}\}$ for $x, y \in Vars(\mathcal{C})$, S needs to be at least of size N since it needs to include all vertices $extra_i$ for $i = 1, \dots, N$;
2. for $G_{\mathcal{C}} \setminus S$ to be a polytree and $|S| \leq N$, the set S needs to include at least N; and
3. for $G_{\mathcal{C}} \setminus S$ to be acyclic and $|S| \leq N$, the set S needs to include v_x or $v_{\neg x}$ for each $x \in Vars(\mathcal{X})$.

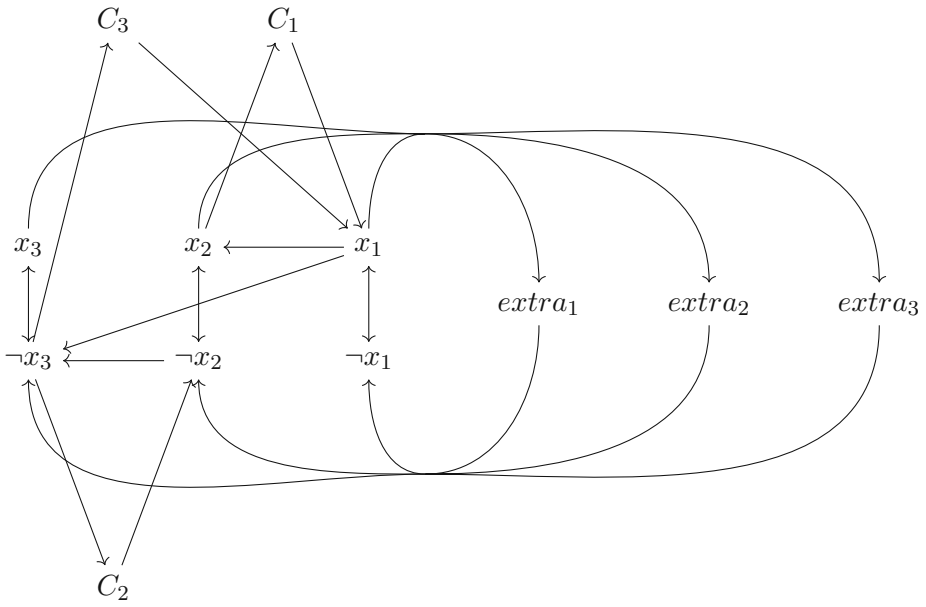


Figure C.1: The digraph $G_{\mathcal{C}}$ constructed in the proof of Theorem 162 for the CNF $\mathcal{C} = \{C_1, C_2, C_3\}$ with $C_1 = x_1 \vee x_2$, $C_2 = \neg x_2 \vee \neg x_3$, and $C_3 = x_1 \vee \neg x_3$. Note that the edges between x_i (resp. $\neg x_i$) and $extra_j$ are by intent drawn in an overlapping manner to improve readability.

This means that we either need to remove a set S such that $G_{\mathcal{C}} \setminus S$ is cyclic but not strongly connected or a set S of size N such that $G_{\mathcal{C}} \setminus S$ is acyclic. The latter would imply that $\mathcal{C}$ is satisfiable and the former is impossible, since due to (1.) we would need S of size at least N to obtain N different SCCs and using a set of size k to obtain less than $k+1$ different SCCs is not helpful, since analogous observations hold for the obtained SCCs. □

APPENDIX D

Implementation Details

D.1 Knowledge Compilation Settings

We use the different solvers for knowledge compilation as follows:

d4 For d4, we simply used standard settings adding only the input arguments "-dDNNF", to ensure that we obtain a d-DNNF as output, "-out=file_name.nnf", to ensure that the d-DNNF is saved in the desired output file, and "-smooth", to activate our minor modification that ensures the resulting d-DNNF is smooth.

sharpSAT-TD For SHARPSAT-TD, we set the argument "-decot", i.e., the time used by SHARPSAT-TD to compute a tree decomposition based on the corresponding input argument to aspmc. The other arguments are set as follows:

- "-decow", specifies the importance of the tree decomposition guidance. We set it to 100.
- "-dDNNF", is a new option that we added, and specifies that we want to use SHARPSAT-TD as a knowledge compiler.
- "-tmpdir", simply specifies the temporary directory that SHARPSAT-TD should use. We use "/tmp/".
- "-cs", denotes the maximum size of the cache in megabyte. We fix it to "3500".
- "-dDNNF_out", specifies the filename that the d-DNNF should be written to. We set it to the name of a new temporary file.

Without the argument "-dDNNF_out" our modified version of sharpSAT-TD prints the d-DNNF it compiles to stdout. This allows us to perform knowledge compilation and evaluation simultaneously. While we implemented this feature in aspmc it should only lead to a speed up by a factor of up to two and is not included in the empirical evaluation.

c2d For c2d, we set the following input arguments:

- "-smooth_all", specifies that the final output d-DNNF should be smooth.
- "-reduce", specifies that c2d should perform possible reduction steps before outputting the final d-DNNF.
- "-dt_in", specifies the input dtree that should be used to guide the variable selection of c2d. While this is an optional argument, we chose to provide a dtree that we generated ourselves.
- "-cache_size", specifies the maximum size of the cache, which we set to "3500".

The other input arguments (apart from the CNF to compile) are not used.

miniC2D For miniC2D, we set the following input arguments:

- "-v", specifies the input vtree that defines the order in which variables are decided by miniC2D. Similarly to the dtree argument of c2d, this argument is optional but we chose to provide a vtree that we generated ourselves.
- "-s", specifies the maximum size of the cache. We set it to "3500".

The other input arguments (apart from CNF to compile) are not used.

D.2 Dtree and Vtree Generation

Algorithm 3 TD_TO_DTREE(C, td)

Input A CNF C and a tree decomposition td of PRIM(C).
Output A dtree for C that corresponds to td.

```
td_node_to_clauses = {} # at which td node the clause should be handles
last_td_node = {} # at which td node each variable occurs last
idx = 0
bag_idx = list(td)
for bag in td.bag_iter() do
    clauses[bag] = []
    bag.idx = idx
    for a in bag.vertices do
        last_td_node = idx
    idx += 1
# set where each clause should be handled
for i,c in enumerate(cnf.clauses) do
    idx = min([ last_td_node[abs(b)] for b in c ])
    td_node_to_clauses[bag_idx[idx]].append(i)
dtree_idx = [ None for _ in range(td.bags) ]
for bag in td.bag_iter() do
    cur_dtree = None
    for child in bag.children do
        child_dtree = dtree_idx[child.idx]
        if cur_dtree == None then
            cur_dtree = child_dtree
        else
            if child_dtree != None then
                cur_dtree = Dtree(right = cur_dtree, left = child_tree)
    for i in td_node_to_clauses[bag] do
        if cur_dtree == None then
            cur_dtree = Dtree(val = i)
        else
            cur_dtree = Dtree(right = cur_dtree, left = Dtree(val = i))
    # remember the final dtree for this bag
    dtree_idx[bag.idx] = cur_dtree
return dtree_idx[td.get_root().idx]
```

We describe shortly how we generate dtrees and vtrees from a tree decomposition in such a way that the variables are decided in order of minimum distance from the root of the tree decomposition. Our strategy only slightly differs from that of Korhonen and Järvisalo [KJ21] [KJ21]. Dtrees describe not the order in which variables are processed

but the order in which clauses are processed, which implicitly specifies multiple orders in which the variables can be processed.

The algorithm to generate a dtree from a tree decomposition td for a CNF C is given in Algorithm 3. The general idea is that each clause of C has to occur exactly once in the dtree. Then for a dtree to correspond to a tree decomposition td, we want to first handle all the clauses whose variables occur in the root of td, remove them from the CNF and recurse for each of the children of the root on the remaining clauses, adding one subdtree for each child. Practically, it is expensive to check for each clause whether it is contained in the current bag of the tree decomposition. Instead, Algorithm 3 first checks (in lines 1-12) for each variable in C, which bag of the tree decomposition is the last in post-order traversal to contain it. This tells us that the index of the last bag in post-order to contain all the variables of a clause c is the minimum index over all variables (line 15). Then we want to handle clause c at this bag (line 16). Finally, we build the dtree from bottom up by traversing the tree decomposition again in post-order. Here, we first combine the dtrees of the children of the current bag (lines 22-27) before we add the clauses that need to be added at the current bag (lines 29-33). After processing the current bag, we store the corresponding dtree in a map from bags to dtrees. Then the final dtree is the one that the root of the tree decomposition maps to.

The algorithm to generate a vtree from a tree decomposition works similarly; the only difference is that we directly work on variables instead of clauses.

List of Figures

List of Tables

List of Algorithms

Bibliography

[06] *ASP Tools*. https://research.ics.aalto.fi/software/asp/lp2sat/. 2006.

[17] *Lark - a parsing toolkit for Python*. https://github.com/lark-parser/lark. 2017.

[AB01] Réka Albert and Albert-László Barabási. „Statistical mechanics of complex networks“. In: *CoRR* cond-mat/0106096 (2001). URL: http://arxiv.org/abs/cond-mat/0106096.

[AF13] Mario Alviano and Wolfgang Faber. „The Complexity Boundary of Answer Set Programming with Generalized Atoms under the FLP Semantics“. In: *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*. Ed. by Pedro Cabalar and Tran Cao Son. Vol. 8148. Lecture Notes in Computer Science. Springer, 2013, pp. 67–72. DOI: 10.1007/978-3-642-40564-8_7. URL: https://doi.org/10.1007/978-3-642-40564-8_7.

[AF18] Mario Alviano and Wolfgang Faber. „Aggregates in Answer Set Programming“. In: *Künstliche Intell.* 32.2-3 (2018), pp. 119–124. DOI: 10.1007/s13218-018-0545-9. URL: https://doi.org/10.1007/s13218-018-0545-9.

[Agu+13] Felicidad Aguado, Pedro Cabalar, Martıń Diéguez, Gilberto Pérez, and Concepción Vidal. „Temporal equilibrium logic: a survey“. In: *Journal of Applied Non-Classical Logics* 23.1-2 (2013), pp. 2–24.

[Agu+17] Felicidad Aguado, Pedro Cabalar, Martıń Diéguez, Gilberto Pérez, and Concepción Vidal. „Temporal equilibrium logic with past operators“. In: *Journal of Applied Non-Classical Logics* 27.3-4 (2017), pp. 161–177.

[Ake78] Sheldon B. Akers. „Binary decision diagrams“. In: *IEEE Transactions on computers* 27.06 (1978), pp. 509–516.

[AL15] Mario Alviano and Nicola Leone. „Complexity and compilation of GZ-aggregates in answer set programming“. In: *Theory Pract. Log. Program.* 15.4-5 (2015), pp. 574–587. DOI: 10.1017/S147106841500023X. URL: https://doi.org/10.1017/S147106841500023X.

[Alf98] Jorge L. Ramírez Alfonsín. „On Variations of the Subset Sum Problem“. In: *Discret. Appl. Math.* 81.1-3 (1998), pp. 1–7. DOI: 10.1016/S0166-218X(96)00105-9. URL: https://doi.org/10.1016/S0166-218X(96)00105-9.

[Ama+20] Antoine Amarilli, Florent Capelli, Mikaël Monet, and Pierre Senellart. „Connecting Knowledge Compilation Classes and Width Parameters“. In: *Theory Comput. Syst.* 64.5 (2020), pp. 861–914. DOI: 10.1007/s00224-019-09930-2. URL: https://doi.org/10.1007/s00224-019-09930-2.

[AMR20] Marcelo Arenas, Martin Mu~ noz, and Cristian Riveros. „Descriptive Complexity for Counting Complexity Classes“. In: *Log. Methods Comput. Sci.* 16.1 (2020). DOI: 10.23638/LMCS-16(1:9)2020. URL: https://doi.org/10.23638/LMCS-16(1:9)2020.

[AMW17] Michael Abseher, Nysret Musliu, and Stefan Woltran. „htd - A Free, Open-Source Framework for (Customized) Tree Decompositions and Beyond“. In: *Integration of AI and OR Techniques in Constraint Programming - 14th International Conference, CPAIOR 2017, Padua, Italy, June 5-8, 2017, Proceedings*. Ed. by Domenico Salvagnin and Michele Lombardi. Vol. 10335. Lecture Notes in Computer Science. Springer, 2017, pp. 376–386. DOI: 10.1007/978-3-319-59776-8_30. URL: https://doi.org/10.1007/978-3-319-59776-8_30.

[AV91] Serge Abiteboul and Victor Vianu. „Generic Computation and Its Complexity“. In: *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, May 5-8, 1991, New Orleans, Louisiana, USA*. Ed. by Cris Koutsougeras and Jeffrey Scott Vitter. ACM, 1991, pp. 209–219. DOI: 10.1145/103418.103444. URL: https://doi.org/10.1145/103418.103444.

[AW93] Eric Allender and Klaus W. Wagner. „Counting Hierarchies: Polynomial Time and Constant Depth Circuits“. In: *Current Trends in Theoretical Computer Science - Essays and Tutorials*. Ed. by Grzegorz Rozenberg and Arto Salomaa. Vol. 40. World Scientific Series in Computer Science. World Scientific, 1993, pp. 469–483. DOI: 10.1142/9789812794499_0035. URL: https://doi.org/10.1142/9789812794499_0035.

[Azi+15] Rehan Abdul Aziz, Geoffrey Chu, Christian J. Muise, and Peter James Stuckey. „Stable Model Counting and Its Application in Probabilistic Logic Programming“. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Ed. by Blai Bonet and Sven Koenig. AAAI Press, 2015, pp. 3468–3474. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9709.

[AZZ17] Vernon Asuncion, Yan Zhang, and Heng Zhang. „Polynomially Bounded Logic Programs with Function Symbols: A New Decidable“. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 2017.

[Bac+20] Fahiem Bacchus, Jeremias Berg, Matti Järvisalo, and Ruben Martins, eds. *MaxSAT Evaluation 2020: Solver and Benchmark Descriptions*. English. Vol. B-2020-2. Department of Computer Science Report Series B. Finland: Department of Computer Science, University of Helsinki, 2020.

[Ban+86] F Bancilhon, D Maier, Y Sagiv, and JD Ullman. „Magic sets and other strange ways to implement logic programs," in proceedings of The ACM Symposium on Principles of Database Systems“. In: *Cambridge, Massachusetts, March* (1986), pp. 24–26.

[BD20] Vaishak Belle and Luc De Raedt. „Semiring programming: A semantic framework for generalized sum product problems“. In: *Int. J. Approx. Reason.* 126 (2020), pp. 181–201. DOI: 10.1016/j.ijar.2020.08.001. URL: https://doi.org/10.1016/j.ijar.2020.08.001.

[BDE16] Harald Beck, Minh Dao-Tran, and Thomas Eiter. „Equivalent Stream Reasoning Programs“. In: *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*. Ed. by Subbarao Kambhampati. IJCAI/AAAI Press, 2016, pp. 929–935. URL: http://www.ijcai.org/Abstract/16/136.

[BDE18] Harald Beck, Minh Dao-Tran, and Thomas Eiter. „LARS: A Logic-based framework for Analytic Reasoning over Streams“. In: *Artif. Intell.* 261 (2018), pp. 16–70. DOI: 10.1016/j.artint.2018.04.003. URL: https://doi.org/10.1016/j.artint.2018.04.003.

[BDP09] Fahiem Bacchus, Shannon Dalmao, and Toniann Pitassi. „Solving #SAT and Bayesian Inference with Backtracking Search“. In: *J. Artif. Intell. Res.* 34 (2009), pp. 391–442. DOI: 10.1613/jair.2648. URL: https://doi.org/10.1613/jair.2648.

[Bec14] Harald Beck. „Towards a Logic-Based Framework for Analyzing Stream Reasoning“. In: *Proceedings of the 3rd International Workshop on Ordering and Reasoning Co-located with the 13th International Semantic Web Conference (ISWC 2014), Riva del Garda, Italy, October 20th, 2014*. Ed. by Irene Celino, Óscar Corcho, Daniele Dell'Aglio, Emanuele Della Valle, Markus Krötzsch, and Stefan Schlobach. Vol. 1303. CEUR Workshop Proceedings. CEUR-WS.org, 2014, pp. 11–22. URL: http://ceur-ws.org/Vol-1303/paper_3.pdf.

[BEF17] Harald Beck, Thomas Eiter, and Christian Folie. „Ticker: A system for incremental ASP-based stream reasoning“. In: *Theory and Practice of Logic Programming* 17.5-6 (2017), pp. 744–763.

[BEK21] Loris Bozzato, Thomas Eiter, and Rafael Kiesel. „Reasoning on Multirelational Contextual Hierarchies via Answer Set Programming with Algebraic Measures“. In: *Theory Pract. Log. Program.* 21.5 (2021), pp. 593–609. DOI: 10.1017/S1471068421000284. URL: https://doi.org/10.1017/S1471068421000284.

[BG81] Charles H. Bennett and John Gill. „Relative to a Random Oracle A, P^A != NP^A != co-NP^A with Probability 1“. In: *SIAM Journal on Computing* 10.1 (1981), pp. 96–113. DOI: 10.1137/0210008. URL: https://doi.org/10.1137/0210008.

[BG92] Richard Beigel and John Gill. „Counting Classes: Thresholds, Parity, Mods, and Fewness“. In: *Theor. Comput. Sci.* 103.1 (1992), pp. 3–23. DOI: 10.1016/0304-3975(92)90084-S. URL: https://doi.org/10.1016/0304-3975(92)90084-S.

[BGR09] Chitta Baral, Michael Gelfond, and J. Nelson Rushton. „Probabilistic reasoning with answer sets“. In: *Theory Pract. Log. Program.* 9.1 (2009), pp. 57–144. DOI: 10.1017/S1471068408003645. URL: https://doi.org/10.1017/S1471068408003645.

[Bis+99] Stefano Bistarelli, Ugo Montanari, Francesca Rossi, Thomas Schiex, Gérard Verfaillie, and Hélène Fargier. „Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison“. In: *Constraints An Int. J.* 4.3 (1999), pp. 199–240. DOI: 10.1023/A:1026441215081. URL: https://doi.org/10.1023/A:1026441215081.

[Bjö11] Magnus Björk. „Successful SAT Encoding Techniques“. In: *J. Satisf. Boolean Model. Comput.* 7.4 (2011), pp. 189–201. DOI: 10.3233/sat190085. URL: https://doi.org/10.3233/sat190085.

[BL10] Michael Bartholomew and Joohyung Lee. „A Decidable Class of Groundable Formulas in the General Theory of Stable Models“. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010.* Ed. by Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczynski. AAAI Press, 2010. URL: http://aaai.org/ocs/index.php/KR/KR2010/paper/view/1375.

[Blo+14] Marjon Blondeel, Steven Schockaert, Dirk Vermeir, and Martine De Cock. „Complexity of fuzzy answer set programming under Łukasiewicz semantics“. In: *Int. J. Approx. Reason.* 55.9 (2014), pp. 1971–2003. DOI: 10.1016/j.ijar.2013.10.011. URL: https://doi.org/10.1016/j.ijar.2013.10.011.

[BLR00] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. „Enhancing Disjunctive Datalog by Constraints“. In: *IEEE Trans. Knowl. Data Eng.* 12.5 (2000), pp. 845–860. DOI: 10.1109/69.877512. URL: https://doi.org/10.1109/69.877512.

[BLR97] Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. „Strong and Weak Constraints in Disjunctive Datalog“. In: *Logic Programming and Nonmonotonic Reasoning, 4th International Conference, LPNMR'97, Dagstuhl Castle, Germany, July 28-31, 1997, Proceedings*. Ed. by Jürgen Dix, Ulrich Furbach, and Anil Nerode. Vol. 1265. Lecture Notes in Computer Science. Springer, 1997, pp. 2–17. DOI: 10.1007/3-540-63255-7_2. URL: https://doi.org/10.1007/3-540-63255-7_2.

[Blu98] Lenore Blum. *Complexity and real computation*. Springer, 1998. ISBN: 0387982817. URL: https://www.worldcat.org/oclc/37004484.

[BMR01] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. „Semiring-based contstraint logic programming: syntax and semantics“. In: *ACM Trans. Program. Lang. Syst.* 23.1 (2001), pp. 1–29. DOI: 10.1145/383721.383725. URL: https://doi.org/10.1145/383721.383725.

[BMR97] Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. „Semiring-based Constraint Logic Programming“. In: *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97, Nagoya, Japan, August 23-29, 1997, 2 Volumes*. Morgan Kaufmann, 1997, pp. 352–357. URL: http://ijcai.org/Proceedings/97-1/Papers/055.pdf.

[Bod98] Hans L Bodlaender. „A partial k-arboretum of graphs with bounded treewidth“. In: *Theoretical computer science* 209.1-2 (1998), pp. 1–45.

[Bre+15] Gerhard Brewka, James P. Delgrande, Javier Romero, and Torsten Schaub. „asprin: Customizing Answer Set Preferences without a Headache“. In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA*. Ed. by Blai Bonet and Sven Koenig. AAAI Press, 2015, pp. 1467–1474. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9535.

[BRS18] Stefano Bistarelli, Fabio Rossi, and Francesco Santini. „A novel weighted defence and its relaxation in abstract argumentation“. In: *Int. J. Approx. Reason.* 92 (2018), pp. 66–86. DOI: 10.1016/j.ijar.2017.10.006. URL: https://doi.org/10.1016/j.ijar.2017.10.006.

[BS17] Simone Bova and Stefan Szeider. „Circuit Treewidth, Sentential Decision, and Query Compilation“. In: *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*. Ed. by Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts. ACM, 2017, pp. 233–246. DOI: 10.1145/3034786.3034787. URL: https://doi.org/10.1145/3034786.3034787.

[BSS89] Lenore Blum, Mike Shub, and Steve Smale. „On a theory of computation and complexity over the real numbers: NP- completeness, recursive functions and universal machines“. In: *Bulletin (New Series) of the American Mathematical*

Society 21.1 (1989), pp. 1–46. DOI: bams/1183555121. URL: https://doi.org/.

[Büc60] J. Richard Büchi. „Weak Second-Order Arithmetic and Finite Automata“. In: *Mathematical Logic Quarterly* 6.1-6 (1960), pp. 66–92. DOI: 0.1002/malq.19600060105.

[Cab+18] Pedro Cabalar, Roland Kaminski, Torsten Schaub, and Anna Schuhmann. „Temporal Answer Set Programming on Finite Traces“. In: *Theory Pract. Log. Program.* 18.3-4 (2018), pp. 406–420. DOI: 10.1017/S1471068418000297. URL: https://doi.org/10.1017/S1471068418000297.

[Cab+19] Pedro Cabalar, Roland Kaminski, Philip Morkisch, and Torsten Schaub. „telingo= ASP+ time“. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 2019, pp. 256–269.

[Cab+20a] Pedro Cabalar, Martin Dieguez, Torsten Schaub, and Anna Schuhmann. „Towards metric temporal answer set programming“. In: *Theory and Practice of Logic Programming* 20.5 (2020), pp. 783–798.

[Cab+20b] Pedro Cabalar, Jorge Fandinno, Torsten Schaub, and Philipp Wanko. „A Uniform Treatment of Aggregates and Constraints in Hybrid ASP“. In: *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*. Ed. by Diego Calvanese, Esra Erdem, and Michael Thielscher. 2020, pp. 193–202. DOI: 10.24963/kr.2020/20. URL: https://doi.org/10.24963/kr.2020/20.

[Cab+20c] Pedro Cabalar, Jorge Fandinno, Torsten Schaub, and Philipp Wanko. „An ASP Semantics for Constraints Involving Conditional Aggregates“. In: *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*. Ed. by Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang. Vol. 325. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 664–671. DOI: 10.3233/FAIA200152. URL: https://doi.org/10.3233/FAIA200152.

[Cab+22] Pedro Cabalar, Martín Diéguez, Torsten Schaub, and Anna Schuhmann. „Metric temporal answer set programming over timed traces“. In: *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer. 2022, pp. 117–130.

[Cab11] Pedro Cabalar. „Functional answer set programming“. In: *Theory Pract. Log. Program.* 11.2-3 (2011), pp. 203–233. DOI: 10.1017/S1471068410000517. URL: https://doi.org/10.1017/S1471068410000517.

[Cal+08a] Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. „Computable Functions in ASP: Theory and Implementation“. In: *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*. Ed. by Maria Garcia de la Banda and Enrico Pontelli. Vol. 5366. Lecture Notes in Computer Science. Springer, 2008, pp. 407–424. DOI: 10.1007/978-3-540-89982-2_37. URL: https://doi.org/10.1007/978-3-540-89982-2_37.

[Cal+08b] Francesco Calimeri, Susanna Cozza, Giovambattista Ianni, and Nicola Leone. „Computable functions in ASP: Theory and implementation“. In: *International Conference on Logic Programming*. Springer. 2008, pp. 407–424.

[Cas+14] Julien Cassaigne, Vesa Halava, Tero Harju, and François Nicolas. „Tighter Undecidability Bounds for Matrix Mortality, Zero-in-the-Corner Problems, and More“. In: *CoRR* abs/1404.0644 (2014). arXiv: 1404.0644. URL: http://arxiv.org/abs/1404.0644.

[CD14] Pedro Cabalar and Martıń Diéguez. „Strong Equivalence of Non-Monotonic Temporal Theories“. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. Ed. by Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter. AAAI Press, 2014. URL: http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/7954.

[CDS96] Marco Cadoli, Francesco M. Donini, and Marco Schaerf. „Is Intractability of Nonmonotonic Reasoning a Real Drawback?“ In: *Artif. Intell.* 88.1-2 (1996), pp. 215–251. DOI: 10.1016/S0004-3702(96)00009-4. URL: https://doi.org/10.1016/S0004-3702(96)00009-4.

[CKS01] Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity classifications of Boolean constraint satisfaction problems*. Vol. 7. SIAM monographs on discrete mathematics and applications. SIAM, 2001. ISBN: 978-0-89871-479-1.

[CM20] Fábio Gagliardi Cozman and Denis Deratani Mauá. „The joy of Probabilistic Answer Set Programming: Semantics, complexity, expressivity, inference“. In: *Int. J. Approx. Reason.* 125 (2020), pp. 218–239. DOI: 10.1016/j.ijar.2020.07.004. URL: https://doi.org/10.1016/j.ijar.2020.07.004.

[CNR22] Angelos Charalambidis, Christos Nomikos, and Panos Rondogiannis. „Strong Equivalence of Logic Programs with Ordered Disjunction: A Logical Perspective“. In: *Theory Pract. Log. Program.* 22.5 (2022), pp. 708–722. DOI: 10.1017/S1471068422000242. URL: https://doi.org/10.1017/S1471068422000242.

[CPV09] Pedro Cabalar, David Pearce, and Agustıń Valverde. „A Revised Concept of Safety for General Answer Set Programs“. In: *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*. Ed. by Esra Erdem, Fangzhen Lin, and Torsten Schaub. Vol. 5753. Lecture Notes in Computer Science. Springer, 2009, pp. 58–70. DOI: 10.1007/978-3-642-04238-6_8. URL: https://doi.org/10.1007/978-3-642-04238-6_8.

[CV07] Pedro Cabalar and Gilberto Pérez Vega. „Temporal Equilibrium Logic: A First Approach“. In: *Computer Aided Systems Theory - EUROCAST 2007, 11th International Conference on Computer Aided Systems Theory, Las Palmas de Gran Canaria, Spain, February 12-16, 2007, Revised Selected Papers*. Ed. by Roberto Moreno-Díáz, Franz Pichler, and Alexis Quesada-Arencibia. Vol. 4739. Lecture Notes in Computer Science. Springer, 2007, pp. 241–248. DOI: 10.1007/978-3-540-75867-9_31. URL: https://doi.org/10.1007/978-3-540-75867-9_31.

[Dan+01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. „Complexity and expressive power of logic programming“. In: *ACM Comput. Surv.* 33.3 (2001), pp. 374–425. DOI: 10.1145/502807.502810. URL: https://doi.org/10.1145/502807.502810.

[Dar+17] Adnan Darwiche, Pierre Marquis, Dan Suciu, and Stefan Szeider. „PySDD“. In: *Recent Trends in Knowledge Compilation (Dagstuhl Seminar 17381)*. Vol. 7(9). 2017, pp. 81–82. DOI: 10.4230/DagRep.7.9.62. URL: https://doi.org/10.4230/DagRep.7.9.62.

[Dar04] Adnan Darwiche. „New Advances in Compiling CNF into Decomposable Negation Normal Form“. In: *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*. Ed. by Ramón López de Mántaras and Lorenza Saitta. IOS Press, 2004, pp. 328–332.

[Dar11] Adnan Darwiche. „SDD: A New Canonical Representation of Propositional Knowledge Bases“. In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*. Ed. by Toby Walsh. IJCAI/AAAI, 2011, pp. 819–826. DOI: 10.5591/978-1-57735-516-8/IJCAI11-143. URL: https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-143.

[dCSW20] Cassio P. de Campos, Georgios Stamoulis, and Dennis Weyland. „A structured view on weighted counting with relations to counting, quantum computation and applications“. In: *Inf. Comput.* 275 (2020), p. 104627. DOI: 10.1016/j.ic.2020.104627. URL: https://doi.org/10.1016/j.ic.2020.104627.

[DD20] Vincent Derkinderen and Luc De Raedt. „Algebraic Circuits for Decision Theoretic Inference and Learning“. In: *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*. Ed. by Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang. Vol. 325. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 2569–2576. DOI: 10.3233/FAIA200392. URL: https://doi.org/10.3233/FAIA200392.

[Del+03] Tina Dell'Armi, Wolfgang Faber, Giuseppe Ielpa, Nicola Leone, and Gerald Pfeifer. „Aggregate Functions in Disjunctive Logic Programming: Semantics, Complexity, and Implementation in DLV“. In: *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*. Ed. by Georg Gottlob and Toby Walsh. Morgan Kaufmann, 2003, pp. 847–852. URL: http://ijcai.org/Proceedings/03/Papers/122.pdf.

[Del+16] Holger Dell, Thore Husfeldt, Bart M. P. Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. „The First Parameterized Algorithms and Computational Experiments Challenge“. In: *11th International Symposium on Parameterized and Exact Computation, IPEC 2016, August 24-26, 2016, Aarhus, Denmark*. Ed. by Jiong Guo and Danny Hermelin. Vol. 63. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 30:1–30:9. DOI: 10.4230/LIPIcs.IPEC.2016.30. URL: https://doi.org/10.4230/LIPIcs.IPEC.2016.30.

[Dem92] Robert Demolombe. „Syntactical Characterization of a Subset of Domain-Independent Formulas“. In: *J. ACM* 39.1 (1992), pp. 71–94. DOI: 10.1145/147508.147520. URL: https://doi.org/10.1145/147508.147520.

[DG07] Manfred Droste and Paul Gastin. „Weighted automata and weighted logics“. In: *Theor. Comput. Sci.* 380.1-2 (2007), pp. 69–86. DOI: 10.1016/j.tcs.2007.02.055. URL: https://doi.org/10.1016/j.tcs.2007.02.055.

[DHK05] Arnaud Durand, Miki Hermann, and Phokion G. Kolaitis. „Subtractive reductions and complete problems for counting complexity classes“. In: *Theor. Comput. Sci.* 340.3 (2005), pp. 496–513. DOI: 10.1016/j.tcs.2005.03.012. URL: https://doi.org/10.1016/j.tcs.2005.03.012.

[DK15] Luc De Raedt and Angelika Kimmig. „Probabilistic (logic) programming concepts“. In: *Mach. Learn.* 100.1 (2015), pp. 5–47. DOI: 10.1007/s10994-015-5494-z. URL: https://doi.org/10.1007/s10994-015-5494-z.

[DKT07] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. „ProbLog: A Probabilistic Prolog and Its Application in Link Discovery“. In: *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*. Ed. by Manuela M. Veloso. 2007, pp. 2462–2467. URL: http://ijcai.org/Proceedings/07/Papers/396.pdf.

[DM02] Adnan Darwiche and Pierre Marquis. „A Knowledge Compilation Map“. In: *J. Artif. Intell. Res.* 17 (2002), pp. 229–264. DOI: 10.1613/jair.989. URL: https://doi.org/10.1613/jair.989.

[dMF13] Eduardo Menezes de Morais and Marcelo Finger. „Probabilistic Answer Set Programming“. In: *Brazilian Conference on Intelligent Systems, BRACIS 2013, Fortaleza, CE, Brazil, 19-24 October, 2013*. IEEE Computer Society, 2013, pp. 150–156. DOI: 10.1109/BRACIS.2013.33. URL: https://doi.org/10.1109/BRACIS.2013.33.

[DPV20] Jeffrey M. Dudek, Vu Phan, and Moshe Y. Vardi. „ADDMC: Weighted Model Counting with Algebraic Decision Diagrams“. In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 1468–1476. URL: https://ojs.aaai.org/index.php/AAAI/article/view/5505.

[EG97] Thomas Eiter and Georg Gottlob. „Expressiveness of stable model semantics for disjunctive logic programs with functions“. In: *The Journal of Logic Programming* 33.2 (1997), pp. 167–178.

[EGL16] Esra Erdem, Michael Gelfond, and Nicola Leone. „Applications of Answer Set Programming“. In: *AI Mag.* 37.3 (2016), pp. 53–68. DOI: 10.1609/aimag.v37i3.2678. URL: https://doi.org/10.1609/aimag.v37i3.2678.

[EGS05] Jason Eisner, Eric Goldlust, and Noah A. Smith. „Compiling Comp Ling: Weighted Dynamic Programming and the Dyna Language“. In: *HLT/EMNLP 2005, Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference, 6-8 October 2005, Vancouver, British Columbia, Canada*. The Association for Computational Linguistics, 2005, pp. 281–290. URL: https://aclanthology.org/H05-1036/.

[EHK21] Thomas Eiter, Markus Hecher, and Rafael Kiesel. „Treewidth-Aware Cycle Breaking for Algebraic Answer Set Counting“. In: *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*. Ed. by Meghyn Bienvenu, Gerhard Lakemeyer, and Esra Erdem. 2021, pp. 269–279. DOI:

10.24963/kr.2021/26. URL: https://doi.org/10.24963/kr.2021/26.

[EHK23] Thomas Eiter, Markus Hecher, and Rafael Kiesel. „aspmc: New Frontiers in Algebraic Answer Set Counting“. In: *to appear* (2023).

[EIK09] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. „Answer set programming: A primer“. In: *Reasoning Web International Summer School*. Springer. 2009, pp. 40–110.

[Eis02] Jason Eisner. „Parameter Estimation for Probabilistic Finite-State Transducers“. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 2002, pp. 1–8. DOI: 10.3115/1073083.1073085. URL: https://aclanthology.org/P02-1001/.

[Eit+04] Thomas Eiter, Wolfgang Faber, Michael Fink, Gerald Pfeifer, and Stefan Woltran. „Complexity of Model Checking and Bounded Predicate Arities for Non-ground Answer Set Programming“. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004*. Ed. by Didier Dubois, Christopher A. Welty, and Mary-Anne Williams. AAAI Press, 2004, pp. 377–387. URL: http://www.aaai.org/Library/KR/2004/kr04-040.php.

[Eit+07] Thomas Eiter, Wolfgang Faber, Michael Fink, and Stefan Woltran. „Complexity results for answer set programming with bounded predicate arities and implications“. In: *Ann. Math. Artif. Intell.* 51.2-4 (2007), pp. 123–165. DOI: 10.1007/s10472-008-9086-5. URL: https://doi.org/10.1007/s10472-008-9086-5.

[Eit+08] Thomas Eiter, Giovambattista Ianni, Thomas Lukasiewicz, Roman Schindlauer, and Hans Tompits. „Combining answer set programming with description logics for the semantic web“. In: *Artificial intelligence* 172.12-13 (2008), pp. 1495–1539.

[Eit+13a] Thomas Eiter, Michael Fink, Thomas Krennwallner, and Christoph Redl. „Liberal Safety Criteria for HEX-Programs“. In: *Twenty-Seventh AAAI Conference (AAAI 2013), July 14–18, 2013, Bellevue, Washington, USA*. Ed. by Marie desJardins and Michael Littman. Bellevue, Washington, USA: AAAI Press, July 2013, pp. 267–275. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6209.

[Eit+13b] Thomas Eiter, Michael Fink, Thomas Krennwallner, and Christoph Redl. „Liberal safety for answer set programs with external sources“. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 27. 2013, pp. 267–275.

[Eit+16] Thomas Eiter, Michael Fink, Thomas Krennwallner, and Christoph Redl. „Domain expansion for ASP-programs with external sources“. In: *Artificial Intelligence* 233 (2016), pp. 84–121.

[EK20a] Thomas Eiter and Rafael Kiesel. „ASP(AC): Answer Set Programming with Algebraic Constraints“. In: *Theory Pract. Log. Program.* 20.6 (2020), pp. 895–910. DOI: 10.1017/S1471068420000393. URL: https://doi.org/10.1017/S1471068420000393.

[EK20b] Thomas Eiter and Rafael Kiesel. „Weighted LARS for Quantitative Stream Reasoning“. In: *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*. Ed. by Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang. Vol. 325. Frontiers in Artificial Intelligence and Applications. IOS Press, 2020, pp. 729–736. DOI: 10.3233/FAIA200160. URL: https://doi.org/10.3233/FAIA200160.

[EK21] Thomas Eiter and Rafael Kiesel. „On the Complexity of Sum-of-Products Problems over Semirings“. In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 6304–6311. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16783.

[EK23] Thomas Eiter and Rafael Kiesel. „Semiring Reasoning Frameworks in AI and Their Computational Complexity“. In: *J. Artif. Intell. Res.* 77 (2023), pp. 207–293. DOI: 10.1613/jair.1.13970. URL: https://doi.org/10.1613/jair.1.13970.

[EOS19] Thomas Eiter, Paul Ogris, and Konstantin Schekotihin. „A distributed approach to LARS stream reasoning (system paper)“. In: *Theory and Practice of Logic Programming* 19.5-6 (2019), pp. 974–989.

[EPS04] Islam Elkabani, Enrico Pontelli, and Tran Cao Son. „Smodels with CLP and Its Applications: A Simple and Effective Approach to Aggregates in ASP“. In: *Logic Programming, 20th International Conference, ICLP 2004, Saint-Malo, France, September 6-10, 2004, Proceedings*. Ed. by Bart Demoen and Vladimir Lifschitz. Vol. 3132. Lecture Notes in Computer Science. Springer, 2004, pp. 73–89. DOI: 10.1007/978-3-540-27775-0_6. URL: https://doi.org/10.1007/978-3-540-27775-0_6.

[ES22] Thomas Eiter and Patrik Schneider. „A Qualitative Temporal Extension of Here-and-There Logic“. In: *Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings*. Ed. by Georg Gottlob, Daniela Inclezan, and Marco Maratea. Vol. 13416. Lecture Notes in Computer Science. Springer, 2022, pp. 159–176. DOI: 10.1007/978-3-031-15707-3_13. URL: https://doi.org/10.1007/978-3-031-15707-3_13.

[Fag74] Ronald Fagin. „Generalized first-order spectra and polynomial-time recognizable sets“. In: *Complexity of computation* 7 (1974), pp. 43–73.

[Fag94] François Fages. „Consistency of Clark's completion and existence of stable models“. In: *Journal of Methods of logic in computer science* 1.1 (1994), pp. 51–60.

[Fal+18] Andreas A. Falkner, Gerhard Friedrich, Konstantin Schekotihin, Richard Taupe, and Erich Christian Teppan. „Industrial Applications of Answer Set Programming“. In: *Künstliche Intell.* 32.2-3 (2018), pp. 165–176. DOI: 10.1007/s13218-018-0548-6. URL: https://doi.org/10.1007/s13218-018-0548-6.

[FD16] Abram L. Friesen and Pedro M. Domingos. „The Sum-Product Theorem: A Foundation for Learning Tractable Models“. In: *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*. Ed. by Maria-Florina Balcan and Kilian Q. Weinberger. Vol. 48. JMLR Workshop and Conference Proceedings. JMLR.org, 2016, pp. 1909–1918. URL: http://proceedings.mlr.press/v48/friesen16.html.

[Fer11] Paolo Ferraris. „Logic programs with propositional connectives and aggregates“. In: *ACM Trans. Comput. Log.* 12.4 (2011), 25:1–25:40. DOI: 10.1145/1970398.1970401. URL: https://doi.org/10.1145/1970398.1970401.

[FFK94] Stephen A. Fenner, Lance Fortnow, and Stuart A. Kurtz. „Gap-Definable Counting Classes“. In: *J. Comput. Syst. Sci.* 48.1 (1994), pp. 116–148. DOI: 10.1016/S0022-0000(05)80024-8. URL: https://doi.org/10.1016/S0022-0000(05)80024-8.

[FFP10] Sergio Flesca, Filippo Furfaro, and Francesco Parisi. „Querying and repairing inconsistent numerical databases“. In: *ACM Trans. Database Syst.* 35.2 (2010), 14:1–14:50. DOI: 10.1145/1735886.1735893. URL: https://doi.org/10.1145/1735886.1735893.

[FHH21] Johannes Klaus Fichte, Markus Hecher, and Florim Hamiti. „The Model Counting Competition 2020“. In: *ACM J. Exp. Algorithmics* 26 (2021), 13:1–13:26. DOI: 10.1145/3459080. URL: https://doi.org/10.1145/3459080.

[Fic+17] Johannes K Fichte, Markus Hecher, Michael Morak, and Stefan Woltran. „DynASP2. 5: Dynamic programming on tree decompositions in action“. In: *International Symposium on Parameterized and Exact Computation (IPEC)*. Vol. 89. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 17:1–17:13.

[Fie+11] Daan Fierens, Guy Van den Broeck, Ingo Thon, Bernd Gutmann, and Luc De Raedt. „Inference in probabilistic logic programs using weighted CNF's“. In: *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*. 2011, pp. 211–220.

[Fie+15] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. „Inference and learning in probabilistic logic programs using weighted Boolean formulas“. In: *Theory Pract. Log. Program.* 15.3 (2015), pp. 358–401. DOI: 10.1017/S1471068414000076. URL: https://doi.org/10.1017/S1471068414000076.

[Fin11] Michael Fink. „A general framework for equivalences in Answer-Set Programming by countermodels in the logic of Here-and-There“. In: *Theory Pract. Log. Program.* 11.2-3 (2011), pp. 171–202. DOI: 10.1017/S1471068410000542. URL: https://doi.org/10.1017/S1471068410000542.

[FL05] Paolo Ferraris and Vladimir Lifschitz. „Weight constraints as nested expressions“. In: *Theory Pract. Log. Program.* 5.1-2 (2005), pp. 45–74. DOI: 10.1017/S1471068403001923. URL: https://doi.org/10.1017/S1471068403001923.

[FL10] Paolo Ferraris and Vladimir Lifschitz. „On the stable model semantics of first-order formulas with aggregates“. In: *Proceedings of International Workshop on Nonmonotonic Reasoning (NMR)*. 2010.

[FLP04] Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. „Recursive Aggregates in Disjunctive Logic Programs: Semantics and Complexity“. In: *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004, Proceedings*. Ed. by José Júlio Alferes and Jo~ ao Alexandre Leite. Vol. 3229. Lecture Notes in Computer Science. Springer, 2004, pp. 200–212. DOI: 10.1007/978-3-540-30227-8_19. URL: https://doi.org/10.1007/978-3-540-30227-8_19.

[FMW19] Wolfgang Faber, Michael Morak, and Stefan Woltran. „Strong Equivalence for Epistemic Logic Programs Made Easy“. In: *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 2809–2816. DOI: 10.1609/aaai.v33i01.33012809. URL: https://doi.org/10.1609/aaai.v33i01.33012809.

[FPL11] Wolfgang Faber, Gerald Pfeifer, and Nicola Leone. „Semantics and complexity of recursive aggregates in answer set programming“. In: *Artif. Intell.* 175.1 (2011), pp. 278–298. DOI: 10.1016/j.artint.2010.04.002. URL: https://doi.org/10.1016/j.artint.2010.04.002.

[FPV05] Andrea Ferrara, Guoqiang Pan, and Moshe Y. Vardi. „Treewidth in Verification: Local vs. Global“. In: *Logic for Programming, Artificial Intelligence, and Reasoning, 12th International Conference, LPAR 2005, Montego Bay, Jamaica, December 2-6, 2005, Proceedings*. Ed. by Geoff Sutcliffe and Andrei Voronkov. Vol. 3835. Lecture Notes in Computer Science. Springer, 2005, pp. 489–503. DOI: 10.1007/11591191_34. URL: https://doi.org/10.1007/11591191_34.

[FS15] Johannes Klaus Fichte and Stefan Szeider. „Backdoors to tractable answer set programming“. In: *Artif. Intell.* 220 (2015), pp. 64–103. DOI: 10.1016/j.artint.2014.12.001. URL: https://doi.org/10.1016/j.artint.2014.12.001.

[Gan+22] Robert Ganian, Eun Jung Kim, Friedrich Slivovsky, and Stefan Szeider. „Sum-of-Products with Default Values: Algorithms and Complexity Results“. In: *J. Artif. Intell. Res.* 73 (2022), pp. 535–552. DOI: 10.1613/jair.1.12370. URL: https://doi.org/10.1613/jair.1.12370.

[Geb+11] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. „Potassco: The Potsdam answer set solving collection“. In: *Ai Communications* 24.2 (2011), pp. 107–124.

[Geb+14] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. „Clingo = ASP + Control: Preliminary Report“. In: *CoRR* abs/1405.3694 (2014). arXiv: 1405.3694. URL: http://arxiv.org/abs/1405.3694.

[Geb+15] Martin Gebser, Amelia Harrison, Roland Kaminski, Vladimir Lifschitz, and Torsten Schaub. „Abstract gringo“. In: *Theory Pract. Log. Program.* 15.4-5 (2015), pp. 449–463. DOI: 10.1017/S1471068415000150. URL: https://doi.org/10.1017/S1471068415000150.

[GHN04] Jiong Guo, Falk Hüffner, and Rolf Niedermeier. „A Structural View on Parameterizing Problems: Distance from Triviality“. In: *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*. Ed. by Rodney G. Downey, Michael R. Fellows, and Frank K. H. A. Dehne. Vol. 3162. Lecture Notes in Computer Science. Springer, 2004, pp. 162–173. DOI: 10.1007/978-3-540-28639-4_15. URL: https://doi.org/10.1007/978-3-540-28639-4_15.

[Gil77] John Gill. „Computational Complexity of Probabilistic Turing Machines“. In: *SIAM J. Comput.* 6.4 (1977), pp. 675–695. DOI: 10.1137/0206049. URL: https://doi.org/10.1137/0206049.

[GJ79] Michael R Gary and David S Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. 1979.

[GKT07] Todd J. Green, Gregory Karvounarakis, and Val Tannen. „Provenance semirings“. In: *Proceedings of the Twenty-Sixth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 11-13, 2007, Beijing, China*. Ed. by Leonid Libkin. ACM, 2007, pp. 31–40. DOI: 10.1145/1265530.1265535. URL: https://doi.org/10.1145/1265530.1265535.

[GL88] Michael Gelfond and Vladimir Lifschitz. „The Stable Model Semantics for Logic Programming“. In: *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, August 15-19, 1988 (2 Volumes)*. Ed. by Robert A. Kowalski and Kenneth A. Bowen. MIT Press, 1988, pp. 1070–1080.

[GM95] Erich Grädel and Klaus Meer. „Descriptive complexity theory over the real numbers“. In: *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 29 May-1 June 1995, Las Vegas, Nevada, USA*. Ed. by Frank Thomson Leighton and Allan Borodin. ACM, 1995, pp. 315–324. DOI: 10.1145/225058.225151. URL: https://doi.org/10.1145/225058.225151.

[GMD13] Laura Giordano, Alberto Martelli, and Daniele Theseider Dupré. „Reasoning about actions with temporal answer sets“. In: *Theory and Practice of Logic Programming* 13.2 (2013), pp. 201–225.

[Goo99] Joshua Goodman. „Semiring Parsing“. In: *Comput. Linguistics* 25.4 (1999), pp. 573–605.

[GP03] Georg Gottlob and Christos H. Papadimitriou. „On the complexity of single-rule datalog queries“. In: *Inf. Comput.* 183.1 (2003), pp. 104–122. DOI: 10.1016/S0890-5401(03)00012-9. URL: https://doi.org/10.1016/S0890-5401(03)00012-9.

[Gre99] Sergio Greco. „Dynamic Programming in Datalog with Aggregates“. In: *IEEE Trans. Knowl. Data Eng.* 11.2 (1999), pp. 265–283. DOI: 10.1109/69.761663. URL: https://doi.org/10.1109/69.761663.

[GT17] Todd J. Green and Val Tannen. „The Semiring Framework for Database Provenance“. In: *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017*. Ed. by Emanuel Sallinger, Jan Van den Bussche, and Floris Geerts. ACM, 2017, pp. 93–99. DOI: 10.1145/3034786.3056125. URL: https://doi.org/10.1145/3034786.3056125.

[Hah+22] Susana Hahn, Tomi Janhunen, Roland Kaminski, Javier Romero, Nicolas Rühling, and Torsten Schaub. „Plingo: A System for Probabilistic Reasoning in Clingo Based on LP $^{\text{MLN}}$ “. In: Lecture Notes in Computer Science 13752 (2022). Ed. by Guido Governatori and Anni-Yasmin Turhan, pp. 54–62. DOI: 10.1007/978-3-031-21541-4_4. URL: https://doi.org/10.1007/978-3-031-21541-4_4.

[Hec22] Markus Hecher. „Treewidth-aware reductions of normal ASP to SAT - Is normal ASP harder than SAT after all?“ In: *Artif. Intell.* 304 (2022), p. 103651. DOI: 10.1016/j.artint.2021.103651. URL: https://doi.org/10.1016/j.artint.2021.103651.

[Her90] Ulrich Hertrampf. „Relations Among Mod-Classes“. In: *Theor. Comput. Sci.* 74.3 (1990), pp. 325–328. DOI: 10.1016/0304-3975(90)90081-R. URL: https://doi.org/10.1016/0304-3975(90)90081-R.

[HS18] Michael Hamann and Ben Strasser. „Graph Bisection with Pareto Optimization“. In: *ACM J. Exp. Algorithmics* 23 (2018). DOI: 10.1145/3173045. URL: https://doi.org/10.1145/3173045.

[HSH12] François Hantry, Lakhdar Saïs, and Mohand-Said Hacid. „On the Complexity of Computing Minimal Unsatisfiable LTL formulas“. In: *CoRR* abs/1203.3706 (2012). arXiv: 1203.3706. URL: http://arxiv.org/abs/1203.3706.

[IP01] Russell Impagliazzo and Ramamohan Paturi. „On the Complexity of k-SAT“. In: *J. Comput. Syst. Sci.* 62.2 (2001), pp. 367–375. DOI: 10.1006/jcss.2000.1727. URL: https://doi.org/10.1006/jcss.2000.1727.

[Jan04] Tomi Janhunen. „Representing Normal Programs with Clauses“. In: *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain, August 22-27, 2004*. Ed. by Ramón López de Mántaras and Lorenza Saitta. IOS Press, 2004, pp. 358–362.

[JN11] Tomi Janhunen and Ilkka Niemelä. „Compact Translations of Non-disjunctive Answer Set Programs to Propositional Clauses“. In: *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*. Ed. by Marcello Balduccini and Tran Cao Son. Vol. 6565. Lecture Notes in Computer Science. Springer, 2011, pp. 111–130. DOI: 10.1007/978-3-642-20832-4_8. URL: https://doi.org/10.1007/978-3-642-20832-4_8.

[Joh21] Markus Hecher Johannes K. Fichte. *Model Counting Competition 2021*. https://mccompetition.org. 2021.

[JPW09] Michael Jakl, Reinhard Pichler, and Stefan Woltran. „Answer-Set Programming with Bounded Treewidth“. In: *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*. Ed. by Craig Boutilier. 2009, pp. 816–822. URL: http://ijcai.org/Proceedings/09/Papers/140.pdf.

[JS12] Abhay Kumar Jha and Dan Suciu. „On the tractability of query compilation and bounded treewidth“. In: *15th International Conference on Database Theory, ICDT '12, Berlin, Germany, March 26-29, 2012*. Ed. by Alin Deutsch. ACM, 2012, pp. 249–261. DOI: 10.1145/2274576.2274603. URL: https://doi.org/10.1145/2274576.2274603.

[JT95] Birgit Jenner and Jacobo Torán. „Computing Functions with Parallel Queries to NP“. In: *Theor. Comput. Sci.* 141.1&2 (1995), pp. 175–193. DOI: 10.1016/0304-3975(94)00080-3. URL: https://doi.org/10.1016/0304-3975(94)00080-3.

[KBvH01] Arie M. C. A. Koster, Hans L. Bodlaender, and Stan P. M. van Hoesel. „Treewidth: Computational Experiments“. In: *Electron. Notes Discret. Math.* 8 (2001), pp. 54–57. DOI: 10.1016/S1571-0653(05)80078-2. URL: https://doi.org/10.1016/S1571-0653(05)80078-2.

[KJ21] Tuukka Korhonen and Matti Järvisalo. „Integrating Tree Decompositions into Decision Heuristics of Propositional Model Counters (Short Paper)“. In: *27th International Conference on Principles and Practice of Constraint Programming, CP 2021, Montpellier, France (Virtual Conference), October 25-29, 2021*. Ed. by Laurent D. Michel. Vol. 210. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 8:1–8:11. DOI: 10.4230/LIPIcs.CP.2021.8. URL: https://doi.org/10.4230/LIPIcs.CP.2021.8.

[KL80] Richard M. Karp and Richard J. Lipton. „Some Connections between Nonuniform and Uniform Complexity Classes“. In: *Proceedings of the 12th Annual ACM Symposium on Theory of Computing, April 28-30, 1980, Los Angeles, California, USA*. Ed. by Raymond E. Miller, Seymour Ginsburg, Walter A. Burkhard, and Richard J. Lipton. ACM, 1980, pp. 302–309. DOI: 10.1145/800141.804678. URL: https://doi.org/10.1145/800141.804678.

[KL82] R. Karp and R. J. Lipton. „Turing machines that take advice“. In: *L'Enseign. Math.* Vol. 28. 1982.

[Klo94] Ton Kloks. *Treewidth, Computations and Approximations*. Vol. 842. Lecture Notes in Computer Science. Springer, 1994. ISBN: 3-540-58356-4. DOI: 10.1007/BFb0045375. URL: https://doi.org/10.1007/BFb0045375.

[KNR16] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. „FAQ: Questions Asked Frequently“. In: *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2016, San Francisco, CA, USA, June 26 - July 01, 2016*. Ed. by Tova Milo and Wang-Chiew Tan. ACM, 2016, pp. 13–28. DOI: 10.1145/2902251.2902280. URL: https://doi.org/10.1145/2902251.2902280.

[KP18a] Krzysztof Kiljan and Marcin Pilipczuk. „Experimental Evaluation of Parameterized Algorithms for Feedback Vertex Set“. In: *17th International Symposium on Experimental Algorithms, SEA 2018, June 27-29, 2018, L'Aquila, Italy*. Ed. by Gianlorenzo D'Angelo. Vol. 103. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018, 12:1–12:12. DOI: 10.4230/LIPIcs.SEA.2018.12. URL: https://doi.org/10.4230/LIPIcs.SEA.2018.12.

[KP18b] Krzysztof Kiljan and Marcin Pilipczuk. *Feedback Vertex Set Experiments*. https://github.com/karek/FeedbackVertexSet-Experiments. 2018.

[Kre88] Mark W. Krentel. „The Complexity of Optimization Problems“. In: *J. Comput. Syst. Sci.* 36.3 (1988), pp. 490–509. DOI: 10.1016/0022-0000(88)90039-6. URL: https://doi.org/10.1016/0022-0000(88)90039-6.

[KS92] Michael Kifer and V. S. Subrahmanian. „Theory of Generalized Annotated Logic Programming and its Applications“. In: *J. Log. Program.* 12.3&4 (1992), pp. 335–367. DOI: 10.1016/0743-1066(92)90007-P. URL: https://doi.org/10.1016/0743-1066(92)90007-P.

[KS93] Ephraim Korach and Nir Solel. „Tree-width, path-width, and cutwidth“. In: *Discrete Applied Mathematics* 43.1 (1993), pp. 97–101.

[KTK22] Rafael Kiesel, Pietro Totis, and Angelika Kimmig. „Efficient Knowledge Compilation Beyond Weighted Model Counting“. In: *Theory Pract. Log. Program.* 22.4 (2022), pp. 505–522. DOI: 10.1017/S147106842200014X. URL: https://doi.org/10.1017/S147106842200014X.

[KVD11] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. „An Algebraic Prolog for Reasoning about Possible Worlds“. In: *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*. Ed. by Wolfram Burgard and Dan Roth. AAAI Press, 2011, pp. 209–214. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI11/paper/view/3685.

[KVD17] Angelika Kimmig, Guy Van den Broeck, and Luc De Raedt. „Algebraic model counting“. In: *J. Appl. Log.* 22 (2017), pp. 46–62. DOI: 10.1016/j.jal.2016.11.031. URL: https://doi.org/10.1016/j.jal.2016.11.031.

[KW08] Jürg Kohlas and Nic Wilson. „Semiring induced valuation algebras: Exact and approximate local computation algorithms“. In: *Artif. Intell.* 172.11 (2008), pp. 1360–1399. DOI: 10.1016/j.artint.2008.03.003. URL: https://doi.org/10.1016/j.artint.2008.03.003.

[Lad89] Richard E. Ladner. „Polynomial Space Counting Problems“. In: *SIAM J. Comput.* 18.6 (1989), pp. 1087–1097. DOI: 10.1137/0218073. URL: https://doi.org/10.1137/0218073.

[Leo+06] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. „The DLV system for knowledge representation and reasoning“. In: *ACM Trans. Comput. Log.* 7.3 (2006), pp. 499–562. DOI: 10.1145/1149114.1149117. URL: https://doi.org/10.1145/1149114.1149117.

[Lie14] Yuliya Lierler. „Relating constraint answer set programming languages and algorithms“. In: *Artif. Intell.* 207 (2014), pp. 1–22. DOI: 10.1016/j.artint.2013.10.004. URL: https://doi.org/10.1016/j.artint.2013.10.004.

[Lie22] Yuliya Lierler. „Strong Equivalence and Program Structure in Arguing Essential Equivalence between Logic Programs“. In: *Theory Pract. Log. Program.* 22.3 (2022), pp. 335–366. DOI: 10.1017/S1471068421000545. URL: https://doi.org/10.1017/S1471068421000545.

[Lif08] Vladimir Lifschitz. „What Is Answer Set Programming?“ In: *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*. Ed. by Dieter Fox and Carla P. Gomes. AAAI Press, 2008, pp. 1594–1597. URL: http://www.aaai.org/Library/AAAI/2008/aaai08-270.php.

[Lif16] Vladimir Lifschitz. „Intelligent Instantiation and Supersafe Rules“. In: *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA*. Ed. by Manuel Carro, Andy King, Neda Saeedloei, and Marina De Vos. Vol. 52. OASIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 7:1–7:14. DOI: 10.4230/OASIcs.ICLP.2016.7. URL: https://doi.org/10.4230/OASIcs.ICLP.2016.7.

[Lif21] Vladimir Lifschitz. „Here and There with Arithmetic“. In: *Theory Pract. Log. Program.* 21.6 (2021), pp. 735–749. DOI: 10.1017/S1471068421000338. URL: https://doi.org/10.1017/S1471068421000338.

[Lin02] Fangzhen Lin. „Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic“. In: *Proceedings of the Eights International Conference on Principles and Knowledge Representation and Reasoning (KR-02), Toulouse, France, April 22-25, 2002*. Ed. by Dieter Fensel, Fausto Giunchiglia, Deborah L. McGuinness, and Mary-Anne Williams. Morgan Kaufmann, 2002, pp. 170–176.

[Lin99] Jørn Lind-Nielsen. *BuDDy: A binary decision diagram package.* http://sourceforge.net/projects/buddy/. 1999.

[LL09] Yuliya Lierler and Vladimir Lifschitz. „One More Decidable Class of Finitely Ground Programs“. In: *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings*. Ed. by Patricia M. Hill and David Scott Warren. Vol. 5649. Lecture Notes in Computer Science. Springer, 2009, pp. 489–493. DOI: 10.1007/978-3-642-02846-5_40. URL: https://doi.org/10.1007/978-3-642-02846-5_40.

[LLP08] Joohyung Lee, Vladimir Lifschitz, and Ravi Palla. „A Reductive Semantics for Counting and Choice in Answer Set Programming.“ In: *AAAI*. 2008, pp. 472–479.

[LM14] Jean-Marie Lagniez and Pierre Marquis. „Preprocessing for Propositional Model Counting“. In: *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada*. Ed. by Carla E. Brodley and Peter Stone. AAAI Press, 2014, pp. 2688–2694. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8264.

[LM17] Jean-Marie Lagniez and Pierre Marquis. „An Improved Decision-DNNF Compiler“. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*. Ed. by Carles Sierra. ijcai.org, 2017, pp. 667–673. DOI: 10.24963/ijcai.2017/93. URL: https://doi.org/10.24963/ijcai.2017/93.

[LOR10] Javier Larrosa, Albert Oliveras, and Enric Rodrıǵuez-Carbonell. „Semiring-Induced Propositional Logic: Definition and Basic Algorithms“. In: *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*. Ed. by Edmund M. Clarke and Andrei Voronkov. Vol. 6355. Lecture Notes in Computer Science. Springer, 2010, pp. 332–347. DOI: 10.1007/978-3-642-17511-4_19. URL: https://doi.org/10.1007/978-3-642-17511-4_19.

[LPV01] Vladimir Lifschitz, David Pearce, and Agustıń Valverde. „Strongly equivalent logic programs“. In: *ACM Trans. Comput. Log.* 2.4 (2001), pp. 526–541. DOI: 10.1145/383779.383783. URL: https://doi.org/10.1145/383779.383783.

[LPV07] Vladimir Lifschitz, David Pearce, and Agustıń Valverde. „A Characterization of Strong Equivalence for Logic Programs with Variables“. In: *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*. Ed. by Chitta Baral, Gerhard Brewka, and John S. Schlipf. Vol. 4483. Lecture Notes in Computer Science. Springer, 2007, pp. 188–200. DOI: 10.1007/978-3-540-72200-7_17. URL: https://doi.org/10.1007/978-3-540-72200-7_17.

[LR06] Vladimir Lifschitz and Alexander A. Razborov. „Why are there so many loop formulas?“ In: *ACM Trans. Comput. Log.* 7.2 (2006), pp. 261–268. DOI: 10.1145/1131313.1131316. URL: https://doi.org/10.1145/1131313.1131316.

[LTT99] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. „Nested Expressions in Logic Programs“. In: *Ann. Math. Artif. Intell.* 25.3-4 (1999), pp. 369–389. DOI: 10.1023/A:1018978005636. URL: https://doi.org/10.1023/A:1018978005636.

[LTW17] Joohyung Lee, Samidh Talsania, and Yi Wang. „Computing LPMLN using ASP and MLN solvers“. In: *Theory Pract. Log. Program.* 17.5-6 (2017), pp. 942–960. DOI: 10.1017/S1471068417000400. URL: https://doi.org/10.1017/S1471068417000400.

[LW16] Joohyung Lee and Yi Wang. „Weighted Rules under the Stable Model Semantics“. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*. Ed. by Chitta Baral, James P. Delgrande, and Frank Wolter. AAAI Press, 2016, pp. 145–154. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12901.

[LY17] Joohyung Lee and Zhun Yang. „LPMLN, Weak Constraints, and P-log“. In: *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA*. Ed. by Satinder Singh and Shaul Markovitch. AAAI Press, 2017, pp. 1170–1177. URL: http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14547.

[LZ03] Fangzhen Lin and Jicheng Zhao. „On Tight Logic Programs and Yet Another Translation from Normal Logic Programs to Propositional Logic“. In: *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*. Ed. by Georg Gottlob and Toby Walsh. Morgan Kaufmann, 2003, pp. 853–858. URL: http://ijcai.org/Proceedings/03/Papers/123.pdf.

[LZ04] Fangzhen Lin and Yuting Zhao. „ASSAT: computing answer sets of a logic program by SAT solvers“. In: *Artif. Intell.* 157.1-2 (2004), pp. 115–137. DOI: 10.1016/j.artint.2004.04.004. URL: https://doi.org/10.1016/j.artint.2004.04.004.

[Man+21] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. „Neural probabilistic logic programming in DeepProbLog“. In: *Artif. Intell.* 298 (2021), p. 103504. DOI: 10.1016/j.artint.2021.103504. URL: https://doi.org/10.1016/j.artint.2021.103504.

[Mat96] Yuri Matiyjasevich. *Hilbert's tenth problem: what can we do with Diophantine equations?* English version of a talk given by the author. Available at http://logic.pdmi.ras.ru/~yumat/personaljournal/H10history/H10histe.pdf. 1996.

[Mee00] Klaus Meer. „Counting problems over the reals“. In: *Theor. Comput. Sci.* 242.1-2 (2000), pp. 41–58. DOI: 10.1016/S0304-3975(98)00190-X. URL: https://doi.org/10.1016/S0304-3975(98)00190-X.

[Mee18] Wannes Meert. *PySDD: Python package for Sentential Decision Diagrams (SDD)*. https://github.com/wannesm/PySDD. 2018.

[Men21] Stefan Mengel. *Counting, Knowledge Compilation and Applications*. 2021. URL: https://tel.archives-ouvertes.fr/tel-03611336.

[MJ10] Theofrastos Mantadelis and Gerda Janssens. „Dedicated Tabling for a Probabilistic Setting“. In: *Technical Communications of the 26th International Conference on Logic Programming, ICLP 2010, July 16-19, 2010, Edinburgh, Scotland, UK*. Ed. by Manuel V. Hermenegildo and Torsten Schaub. Vol. 7. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2010, pp. 124–133. DOI: 10.4230/LIPIcs.ICLP.2010.124. URL: https://doi.org/10.4230/LIPIcs.ICLP.2010.124.

[MR15] Eleni Mandrali and George Rahonis. „Weighted First-Order Logics over Semirings“. In: *Acta Cybern.* 22.2 (2015), pp. 435–483. DOI: 10.14232/actacyb.22.2.2015.13. URL: https://doi.org/10.14232/actacyb.22.2.2015.13.

[Mui+12] Christian Muise, Sheila A McIlraith, J Christopher Beck, and Eric I Hsu. „Dsharp: fast d-DNNF compilation with sharpSAT“. In: *Canadian Conference on Artificial Intelligence*. Springer. 2012, pp. 356–361.

[Nie08] Ilkka Niemelä. „Stable models and difference logic“. In: *Ann. Math. Artif. Intell.* 53.1-4 (2008), pp. 313–329. DOI: 10.1007/s10472-009-9118-9. URL: https://doi.org/10.1007/s10472-009-9118-9.

[NL15] Juan Carlos Nieves and Helena Lindgren. „Possibilistic nested logic programs and strong equivalence“. In: *Int. J. Approx. Reason.* 59 (2015), pp. 1–19. DOI: 10.1016/j.ijar.2015.01.004. URL: https://doi.org/10.1016/j.ijar.2015.01.004.

[NM14] Matthias Nickles and Alessandra Mileo. „Web Stream Reasoning Using Probabilistic Answer Set Programming“. In: *Web Reasoning and Rule Systems - 8th International Conference, RR 2014, Athens, Greece, September 15-17, 2014. Proceedings*. Ed. by Roman Kontchakov and Marie-Laure Mugnier. Vol. 8741. Lecture Notes in Computer Science. Springer, 2014, pp. 197–205. DOI: 10.1007/978-3-319-11113-1_16. URL: https://doi.org/10.1007/978-3-319-11113-1_16.

[NM15] Matthias Nickles and Alessandra Mileo. „A System for Probabilistic Inductive Answer Set Programming“. In: *Scalable Uncertainty Management - 9th International Conference, SUM 2015, Québec City, QC, Canada, September 16-18, 2015. Proceedings*. Ed. by Christoph Beierle and Alex Dekhtyar. Vol. 9310. Lecture Notes in Computer Science. Springer, 2015, pp. 99–105. DOI: 10.1007/978-3-319-23540-0_7. URL: https://doi.org/10.1007/978-3-319-23540-0_7.

[NSS99] Ilkka Niemelä, Patrik Simons, and Timo Soininen. „Stable Model Semantics of Weight Constraint Rules“. In: *Logic Programming and Nonmonotonic Reasoning, 5th International Conference, LPNMR'99, El Paso, Texas, USA, December 2-4, 1999, Proceedings*. Ed. by Michael Gelfond, Nicola Leone, and Gerald Pfeifer. Vol. 1730. Lecture Notes in Computer Science. Springer,

1999, pp. 317–331. DOI: 10.1007/3-540-46767-X_23. URL: https://doi.org/10.1007/3-540-46767-X_23.

[OCD16a] Umut Oztok, Arthur Choi, and Adnan Darwiche. „Solving PP^{PP}-Complete Problems Using Knowledge Compilation“. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*. Ed. by Chitta Baral, James P. Delgrande, and Frank Wolter. AAAI Press, 2016, pp. 94–103. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12910.

[OCD16b] Umut Oztok, Arthur Choi, and Adnan Darwiche. „Solving PP^{PP}-Complete Problems Using Knowledge Compilation“. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifteenth International Conference, KR 2016, Cape Town, South Africa, April 25-29, 2016*. Ed. by Chitta Baral, James P. Delgrande, and Frank Wolter. AAAI Press, 2016, pp. 94–103. URL: http://www.aaai.org/ocs/index.php/KR/KR16/paper/view/12910.

[OD14] Umut Oztok and Adnan Darwiche. „On Compiling CNF into Decision-DNNF“. In: *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*. Ed. by Barry O'Sullivan. Vol. 8656. Lecture Notes in Computer Science. Springer, 2014, pp. 42–57. DOI: 10.1007/978-3-319-10428-7_7. URL: https://doi.org/10.1007/978-3-319-10428-7_7.

[OD15] Umut Oztok and Adnan Darwiche. „A Top-Down Compiler for Sentential Decision Diagrams“. In: *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. Ed. by Qiang Yang and Michael J. Wooldridge. AAAI Press, 2015, pp. 3141–3148. URL: http://ijcai.org/Abstract/15/443.

[PE20] Danh Le Phuoc and Thomas Eiter. „An Adaptive Semantic Stream Reasoning Framework for Deep Neural Networks“. In: *Proceedings of the CIKM 2020 Workshops co-located with 29th ACM International Conference on Information and Knowledge Management (CIKM 2020), Galway, Ireland, October 19-23, 2020*. Ed. by Stefan Conrad and Ilaria Tiddi. Vol. 2699. CEUR Workshop Proceedings. CEUR-WS.org, 2020. URL: http://ceur-ws.org/Vol-2699/paper09.pdf.

[Pea06] David Pearce. „Equilibrium logic“. In: *Ann. Math. Artif. Intell.* 47.1-2 (2006), pp. 3–41. DOI: 10.1007/s10472-006-9028-z. URL: https://doi.org/10.1007/s10472-006-9028-z.

[PET21] Danh Le Phuoc, Thomas Eiter, and Anh Lê Tuán. „A Scalable Reasoning and Learning Approach for Neural-Symbolic Stream Fusion“. In: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*. AAAI Press, 2021, pp. 4996–5005. URL: https://ojs.aaai.org/index.php/AAAI/article/view/16633.

[PV04] David Pearce and Agustıń Valverde. „Towards a First Order Equilibrium Logic for Nonmonotonic Reasoning“. In: *Logics in Artificial Intelligence, 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004, Proceedings*. Ed. by José Júlio Alferes and Jo~ ao Alexandre Leite. Vol. 3229. Lecture Notes in Computer Science. Springer, 2004, pp. 147–160. DOI: 10.1007/978-3-540-30227-8_15. URL: https://doi.org/10.1007/978-3-540-30227-8_15.

[PV06] David Pearce and Agustin Valverde. „Quantified Equilibrium Logic and the First Order Logic of Here-and-There“. In: *Technical Report MA-06-02* (2006).

[PV08] David Pearce and Agustıń Valverde. „Quantified Equilibrium Logic and Foundations for Answer Set Programs“. In: *Logic Programming, 24th International Conference, ICLP 2008, Udine, Italy, December 9-13 2008, Proceedings*. Ed. by Maria Garcia de la Banda and Enrico Pontelli. Vol. 5366. Lecture Notes in Computer Science. Springer, 2008, pp. 546–560. DOI: 10.1007/978-3-540-89982-2_46. URL: https://doi.org/10.1007/978-3-540-89982-2_46.

[RC22] Victor Hugo Nascimento Rocha and Fábio Gagliardi Cozman. „A Credal Least Undefined Stable Semantics for Probabilistic Logic Programs and Probabilistic Argumentation“. In: *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel. July 31 - August 5, 2022*. Ed. by Gabriele Kern-Isberner, Gerhard Lakemeyer, and Thomas Meyer. 2022, pp. 309–319. URL: https://proceedings.kr.org/2022/31/.

[Red17] Christoph Redl. „Extending answer set programs with interpreted functions as first-class citizens“. In: *International Symposium on Practical Aspects of Declarative Languages*. Springer. 2017, pp. 68–85.

[Ron+18] Alessandro Ronca, Mark Kaminski, Bernardo Cuenca Grau, Boris Motik, and Ian Horrocks. „Stream Reasoning in Temporal Datalog“. In: *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*. Ed. by Sheila

A. McIlraith and Kilian Q. Weinberger. AAAI Press, 2018, pp. 1941–1948. URL: https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16182.

[RS11] Fabrizio Riguzzi and Terrance Swift. „The PITA System for Logical-Probabilistic Inference“. In: *Latest Advances in Inductive Logic Programming, ILP 2011, Late Breaking Papers, Windsor Great Park, UK, July 31 - August 3, 2011*. Ed. by Stephen H. Muggleton and Hiroaki Watanabe. Imperial College Press / World Scientific, 2011, pp. 79–86. DOI: 10.1142/9781783265091_0010. URL: https://doi.org/10.1142/9781783265091_0010.

[RS18] Fabrizio Riguzzi and Theresa Swift. „A survey of probabilistic logic programming“. In: *Declarative Logic Programming: Theory, Systems, and Applications*. Ed. by Michael Kifer and Yanhong Annie Liu. ACM / Morgan & Claypool, 2018, pp. 185–228. DOI: 10.1145/3191315.3191319. URL: https://doi.org/10.1145/3191315.3191319.

[Sht+14] Dimitar Shterionov, Joris Renkens, Jonas Vlasselaer, Angelika Kimmig, Wannes Meert, and Gerda Janssens. „The Most Probable Explanation for Probabilistic Logic Programs with Annotated Disjunctions“. In: *Inductive Logic Programming - 24th International Conference, ILP 2014, Nancy, France, September 14-16, 2014, Revised Selected Papers*. Ed. by Jesse Davis and Jan Ramon. Vol. 9046. Lecture Notes in Computer Science. Springer, 2014, pp. 139–153. DOI: 10.1007/978-3-319-23708-4_10. URL: https://doi.org/10.1007/978-3-319-23708-4_10.

[Sht15] Dimitar Shterionov. „Design and Development of Probabilistic Inference Pipelines“. PhD thesis. KU Leuven, 2015.

[SI96] Richard Edwin Stearns and Harry B. Hunt III. „An Algebraic Model for Combinatorial Problems“. In: *SIAM J. Comput.* 25.2 (1996), pp. 448–476. DOI: 10.1137/S0097539793243004. URL: https://doi.org/10.1137/S0097539793243004.

[Skr+22] Arseny Skryagin, Wolfgang Stammer, Daniel Ochs, Devendra Singh Dhami, and Kristian Kersting. „Neural-Probabilistic Answer Set Programming“. In: *Proceedings of the 19th International Conference on Principles of Knowledge Representation and Reasoning, KR 2022, Haifa, Israel. July 31 - August 5, 2022*. Ed. by Gabriele Kern-Isberner, Gerhard Lakemeyer, and Thomas Meyer. 2022, pp. 463–473. URL: https://proceedings.kr.org/2022/48/.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. „Word Problems Requiring Exponential Time: Preliminary Report“. In: *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*. Ed. by Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond

Strong. ACM, 1973, pp. 1–9. DOI: 10.1145/800125.804029. URL: https://doi.org/10.1145/800125.804029.

[SNS02] Patrik Simons, Ilkka Niemelä, and Timo Soininen. „Extending and implementing the stable model semantics“. In: *Artif. Intell.* 138.1-2 (2002), pp. 181–234. DOI: 10.1016/S0004-3702(02)00187-X. URL: https://doi.org/10.1016/S0004-3702(02)00187-X.

[Som12] Fabio Somenzi. „CUDD: CU decision diagram package release 2.5.0“. In: *University of Colorado at Boulder* (2012).

[SS16] Amit Sharma and Sunil Kr Singh. „One Way Functions–Conjecture, Status, Applications and Future Research Scope“. In: *International Journal of Computer Applications* 153.8 (2016).

[SS20] Friedrich Slivovsky and Stefan Szeider. „A Faster Algorithm for Propositional Model Counting Parameterized by Incidence Treewidth“. In: *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*. Ed. by Luca Pulina and Martina Seidl. Vol. 12178. Lecture Notes in Computer Science. Springer, 2020, pp. 267–276. DOI: 10.1007/978-3-030-51825-7_19. URL: https://doi.org/10.1007/978-3-030-51825-7_19.

[Str17] Ben Strasser. „Computing Tree Decompositions with FlowCutter: PACE 2017 Submission“. In: *CoRR* abs/1709.08949 (2017). arXiv: 1709.08949. URL: http://arxiv.org/abs/1709.08949.

[Tai95] SATO Taisuke. „A statistical learning method for logic programs with distribution semantics“. In: *Proceedings of the 12th international conference on logic programming*. Citeseer. 1995, pp. 715–729.

[Tam19] Hisao Tamaki. „Positive-instance driven dynamic programming for treewidth“. In: *J. Comb. Optim.* 37.4 (2019), pp. 1283–1311. DOI: 10.1007/s10878-018-0353-z. URL: https://doi.org/10.1007/s10878-018-0353-z.

[TGK20] Efthymia Tsamoura, Vıćtor Gutiérrez-Basulto, and Angelika Kimmig. „Beyond the Grounding Bottleneck: Datalog Techniques for Inference in Probabilistic Logic Programs“. In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 2020, pp. 10284–10291. URL: https://ojs.aaai.org/index.php/AAAI/article/view/6591.

[TKR23] Pietro Totis, Angelika Kimmig, and Luc De Raedt. „smProbLog: Stable Model Semantics in ProbLog for Probabilistic Argumentation“. In: (2023). arXiv: 2304.00879 [cs.AI].

[Tod89] Seinosuke Toda. „On the Computational Power of PP and +P“. In: *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*. IEEE Computer Society, 1989, pp. 514–519. DOI: 10.1109/SFCS.1989.63527. URL: https://doi.org/10.1109/SFCS.1989.63527.

[Tse83] G. S. Tseitin. „On the Complexity of Derivation in Propositional Calculus“. In: *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983, pp. 466–483. ISBN: 978-3-642-81955-1. DOI: 10.1007/978-3-642-81955-1_28. URL: https://doi.org/10.1007/978-3-642-81955-1_28.

[Val79] Leslie G. Valiant. „The Complexity of Enumeration and Reliability Problems“. In: *SIAM J. Comput.* 8.3 (1979), pp. 410–421. DOI: 10.1137/0208032. URL: https://doi.org/10.1137/0208032.

[Van+10] Guy Van den Broeck, Ingo Thon, Martijn van Otterlo, and Luc De Raedt. „DTProbLog: A Decision-Theoretic Probabilistic Prolog“. In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. Ed. by Maria Fox and David Poole. AAAI Press, 2010, pp. 1217–1222. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI10/paper/view/1695.

[VDB09] Joost Vennekens, Marc Denecker, and Maurice Bruynooghe. „CP-logic: A language of causal probabilistic events and its relation to logic programming“. In: *Theory Pract. Log. Program.* 9.3 (2009), pp. 245–308. DOI: 10.1017/S1471068409003767. URL: https://doi.org/10.1017/S1471068409003767.

[vEK76] Maarten H. van Emden and Robert A. Kowalski. „The Semantics of Predicate Logic as a Programming Language“. In: *J. ACM* 23.4 (1976), pp. 733–742. DOI: 10.1145/321978.321991. URL: https://doi.org/10.1145/321978.321991.

[Vla+14] Jonas Vlasselaer, Joris Renkens, Guy Van den Broeck, and Luc De Raedt. „Compiling probabilistic logic programs into sentential decision diagrams“. In: *Proceedings Workshop on Probabilistic Logic Programming (PLP)*. 2014, pp. 1–10.

[Vla+16] Jonas Vlasselaer, Guy Van den Broeck, Angelika Kimmig, Wannes Meert, and Luc De Raedt. „Tp-compilation for inference in probabilistic logic programs“. In: *International Journal of Approximate Reasoning* 78 (2016), pp. 15–32.

[VMD14] Guy Van den Broeck, Wannes Meert, and Adnan Darwiche. „Skolemization for Weighted First-Order Model Counting“. In: *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourteenth International Conference, KR 2014, Vienna, Austria, July 20-24, 2014*. Ed. by Chitta Baral, Giuseppe De Giacomo, and Thomas Eiter. AAAI Press, 2014. URL:

http://www.aaai.org/ocs/index.php/KR/KR14/paper/view/8012.

[VV86] Leslie G. Valiant and Vijay V. Vazirani. „NP is as Easy as Detecting Unique Solutions“. In: *Theor. Comput. Sci.* 47.3 (1986), pp. 85–93. DOI: 10.1016/0304-3975(86)90135-0. URL: https://doi.org/10.1016/0304-3975(86)90135-0.

[Wag86] Klaus W. Wagner. „The Complexity of Combinatorial Problems with Succinct Input Representation“. In: *Acta Informatica* 23.3 (1986), pp. 325–356. DOI: 10.1007/BF00289117. URL: https://doi.org/10.1007/BF00289117.

[Wal+19] Przemyslaw Andrzej Walega, Bernardo Cuenca Grau, Mark Kaminski, and Egor V. Kostylev. „DatalogMTL: Computational Complexity and Expressive Power“. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*. Ed. by Sarit Kraus. ijcai.org, 2019, pp. 1886–1892. DOI: 10.24963/ijcai.2019/261. URL: https://doi.org/10.24963/ijcai.2019/261.

[Wal+21] Przemyslaw Andrzej Walega, David J. Tena Cucala, Egor V. Kostylev, and Bernardo Cuenca Grau. „DatalogMTL with Negation Under Stable Models Semantics“. In: *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, Online event, November 3-12, 2021*. Ed. by Meghyn Bienvenu, Gerhard Lakemeyer, and Esra Erdem. 2021, pp. 609–618. DOI: 10.24963/kr.2021/58. URL: https://doi.org/10.24963/kr.2021/58.

[Wan+21] Bin Wang, Jun Shen, Shutao Zhang, and Zhizheng Zhang. „On the Strong Equivalences for LPMLN Programs“. In: *Log. Methods Comput. Sci.* 17.1 (2021). URL: https://lmcs.episciences.org/7122.

[YIL20] Zhun Yang, Adam Ishay, and Joohyung Lee. „NeurASP: Embracing Neural Networks into Answer Set Programming“. In: *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*. Ed. by Christian Bessiere. ijcai.org, 2020, pp. 1755–1762. DOI: 10.24963/ijcai.2020/243. URL: https://doi.org/10.24963/ijcai.2020/243.