TU WIEN Informatics

A Novel Method for Grounding in Answer-Set Programming

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Viktor Besin
Matrikelnummer 11775831

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Woltran
Mitwirkung: Dipl.-Ing. Dr.techn. Markus Hecher, BSc

Wien, 26. Jänner 2023

Viktor Besin	Stefan Woltran

TU WIEN Informatics

A Novel Method for Grounding in Answer-Set Programming

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Viktor Besin
Registration Number 11775831

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Woltran
Assistance: Dipl.-Ing. Dr.techn. Markus Hecher, BSc

Vienna, 26th January, 2023

Viktor Besin	Stefan Woltran

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

Erklärung zur Verfassung der Arbeit

Viktor Besin

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, 26. Jänner 2023

Viktor Besin

To my grandfather, in loving memory.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my advisors Stefan Woltran and Markus Hecher, without whose guidance and support this thesis would not have been possible. I would also like to take this opportunity to thank them for offering me to work on the latest research topics. In hindsight, it truly amazes me how easy they enthused me with my nemesis-topic Logic.

Furthermore, I would like to thank all my friends and colleagues, whose moral support made the past years much more enjoyable and kept me sane throughout my studies.

Last but not least, I would like to thank my family. Besides the obvious financial support, it was their personal support and faith what kept me going. Thank you for always being willing to listen to my problems during difficult times. I cannot express how lucky I am to have you in my life.

Kurzfassung

Antwortmengenprogrammierung (ASP) ist ein deklaratives Programmierparadigma, das zusammen mit vielen Erweiterungen zur Lösung einer Vielzahl von Problemen in der künstlichen Intelligenz, insbesondere im Bereich der Wissensrepräsentation und Argumentation, weit verbreitet ist. Eine der wichtigsten Herausforderungen bei ASP ist der Prozess der Grundierung, bei dem eine Problemspezifikation in eine Menge logischer Regeln umgewandelt wird, indem Variablen durch Konstanten ersetzt werden. Die Grundierung ist entscheidend für die Gesamtleistung von ASP-Systemen, da sie sich direkt auf die Größe und Komplexität der resultierenden Regelmenge auswirkt und bei bestimmten Problemen zum bekannten ASP-Grundierungsengpass führt, bei dem das grundierte Programm zu groß ist um von einem ASP-Löser verarbeitet zu werden. In dieser Arbeit stellen wir eine neuartige Methode zur Grundierung in ASP vor, die nicht-grundierte Atome in Regeln entkoppelt, um die Auswertung von Regelkörpern an den Lösungsprozess zu delegieren. Zu diesem Zweck präsentieren wir eine Übersetzungen von nicht-grundierten, normalen (strengen) Programmen in grundierte, disjunktive Programme sowie nicht-grundierte, disjunktive zu grundierten, erkenntnistheoretischen Programmen. Im Vergleich zu herkömmlichen Grundierungssystemen liefern unsere Übersetzungen Programme, die nur exponentiell in der maximalen Prädikatenanzahl, und deshalb nur polynomial, wenn diese durch eine Konstante begrenzt ist. Mit der Implementierung eines Prototyps demonstrieren wir die technische Machbarkeit dieser neuen Methode und vergleichen sie mit modernster ASP-Technologie in Bezug auf Grundierungsgröße, Grundierungszeit und Gesamtlaufzeit. Es stellt sich heraus, dass unser Ansatz konkurrenzfähig mit bestehenden Systemen ist.

Abstract

Answer-set programming (Asp) is a declarative programming paradigm that has gained widespread popularity together with many extensions for solving a variety of problems in artificial intelligence, especially in the field of knowledge representation and reasoning. One of the key challenges in Asp is the process of grounding, which involves transforming a high-level problem specification into a set of low-level logical rules by replacing variables with constants. Grounding is crucial for the overall performance of Asp systems, as it directly affects the size and complexity of the resulting rule set, and for certain problems results in the well-known Asp grounding bottleneck, where the ground program is too huge to be processed by the Asp solver. In this thesis, we present a novel method for grounding in Asp which decouples non-ground atoms in rules in order to delegate the evaluation of rule bodies to the solving process. To this end, we present translations from non-ground, normal (tight) programs to ground, disjunctive programs as well as non-ground, disjunctive to ground epistemic logic programs. In comparison to traditional grounding systems, our translations yield programs that are exponential only in the maximum predicate arity, and thus polynomial if this arity is bounded by a constant. With the implementation of a prototype, we demonstrate technical feasibility of this new method and compare to state-of-the-art Asp technology in terms of grounding size, grounding time and total runtime. It turns out that our approach is competitive with existing systems.

Contents

CHAPTER 1

Introduction

Motivation

The last decades have been marked by some major advances in efficiently solving hard problems. To this end, a problem's hardness is typically classified by computational complexity theory and the *polynomial hierarchy*. One of the most crucial problems is *Boolean Satisfiabilty* (SAT), the problem of deciding whether a Boolean formula is true, which is known to be hard for the well-studied problem class NP. While problems that are hard for this class can probably not be solved in polynomial time, the active research evolved efficient decision procedures which led to the development of available and efficient solvers [AS09, Bie08]. However, there was still a notable shift towards studying, solving and applying even harder problems [KL99, Pap94, SM73].

One of them is *Answer Set Programming* (ASP) [BET11, GKKS19, JN16, SW18]. ASP is a well-known problem modeling and solving framework, which, based on the stable model semantics [GL91], not only asks for the satisfiabilty, but also asks for a justification for every variable that is stated to be true. This and other formalisms (see e.g., QSAT [BHvMW09, KL99], #SAT [GSS21]) are known to be main drivers for knowledge representation and reasoning, as well as artificial intelligence. With its capabilites ASP can be seen as an extension of SAT, where knowledge is modeled by the means of rules forming a logic program, a rule-based language whose solutions are sets of atoms, called answer sets. Similar to SAT, the developement of efficient ASP solvers [GKKS19, CPZ19] enables the broad use in not only academic, e.g. in natural language understanding [CRR19], but also industrial applications [FFS+18]. Beyond that, there are even harder extensions to ASP, e.g. the epistemic extension *Epistemic Logic Programming* (ELP) [GL91, Tru11, SE16], which allows for the reasoning over multiple worlds and has been gaining popularity in the early past. Generally, this makes the improvement of ASP systems an important and interesting research topic in all its fields.

Example 1.1 (Asp Capabilities)**.** *Asp can be used to encode many problems effortlessly. Assume the problem of choosing relationships among entities such that the structure does not contain a triangle. This can easily be represented as a graph consisting of vertices and edges, and encoded in the following (non-ground) program.*

Program Π *decides in Rule (1.1) for each edge (e) of a given graph, whether to pick it (p) or not* ($\bar{p}$)*. Then, in Rule (1.2) it is ensured that the choice of edges does not form triangles.*

$$p(A,B) \vee \bar{p}(A,B) \leftarrow e(A,B). \tag{1.1}$$

$$\leftarrow p(X,Y), p(Y,Z), p(X,Z), X \neq Y, Y \neq Z, X \neq Z. \tag{1.2}$$

When instantiating the program with an instance $\{e(1,2).\ e(1,3).\ e(2,3)\}$*, an Asp solver can decide whether an answer exists and enumerate these (see Figure 1.1).*

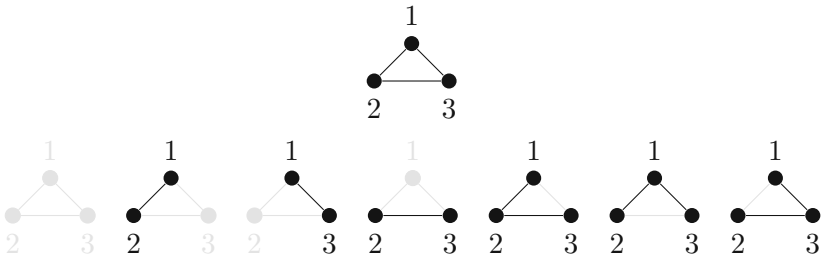

Figure 1.1: Exemplary instance (top) for Example 1.1, where every graph (bottom) depicts a solution (answer set) to the edge-picking problem.

As implied, non-ground programs containing variables require grounding (i.e. the *instantiation* of variables necessary for solving) before solving. Traditional solvers therefore rely on a so-called *ground-and-solve* technique [KLPS16], where a grounding module transforms the non-ground input program into its propositional counterpart by replacing variables with constants and a solving module computes the stable models accordingly. This approach, however, has its limitations for certain problem types. This applies especially for problems where the grounding leads to a combinatorial blow up, i.e. the instantiation of rules yields exponentially large programs whose evaluation is not processable by the solver. This problem is usually referred to as *ASP grounding bottleneck* (cf. Example 1.2).

Example 1.2 (Asp Grounding Bottleneck)**.** *Recall the non-ground program* Π *of Example 1.1.*

One can identify the variables X, Y *and* Z *that are instantiated during grounding. Traditional grounding generates a ground rule for every possible variable allocation from the program's domain* $\mathrm{dom}(\Pi)$ *as follows.*

$$p(1,1) \vee \bar{p}(1,1) \leftarrow e(1,1).\ p(1,2) \vee \bar{p}(1,2) \leftarrow e(1,2).\ p(1,3) \vee \bar{p}(1,3) \leftarrow e(1,3). \tag{1.1}$$
$$p(2,1) \vee \bar{p}(2,1) \leftarrow e(2,1).\ p(2,2) \vee \bar{p}(2,2) \leftarrow e(2,2).\ p(1,3) \vee \bar{p}(2,3) \leftarrow e(2,3).$$
$$p(3,1) \vee \bar{p}(3,1) \leftarrow e(3,1).\ p(3,2) \vee \bar{p}(3,2) \leftarrow e(3,2).\ p(3,3) \vee \bar{p}(3,3) \leftarrow e(3,3).$$

$$\leftarrow p(1,1), p(1,1), p(1,1), 1 \neq 1, 1 \neq 1, 1 \neq 1. \quad \leftarrow p(1,1), p(1,2), p(1,2), 1 \neq 1, 1 \neq 2, 1 \neq 2. \tag{1.2}$$
$$\leftarrow p(1,1), p(1,3), p(1,3), 1 \neq 1, 1 \neq 3, 1 \neq 3. \quad \leftarrow p(1,2), p(2,1), p(1,1), 1 \neq 2, 2 \neq 1, 1 \neq 1.$$
$$\leftarrow p(1,2), p(2,2), p(1,2), 1 \neq 2, 2 \neq 2, 1 \neq 2. \quad \leftarrow p(1,2), p(2,3), p(1,3), 1 \neq 2, 2 \neq 3, 1 \neq 3.$$
$$\leftarrow p(1,3), p(3,1), p(1,1), 1 \neq 3, 3 \neq 1, 1 \neq 1. \quad \leftarrow p(1,3), p(3,2), p(1,2), 1 \neq 3, 3 \neq 2, 1 \neq 2.$$
$$\leftarrow p(1,3), p(3,3), p(1,3), 1 \neq 3, 3 \neq 3, 1 \neq 3. \quad \leftarrow p(2,1), p(1,1), p(2,1), 2 \neq 1, 1 \neq 1, 2 \neq 1.$$
$$\leftarrow p(2,1), p(1,2), p(2,2), 2 \neq 1, 1 \neq 2, 2 \neq 2. \quad \leftarrow p(2,1), p(1,3), p(2,3), 2 \neq 1, 1 \neq 3, 2 \neq 3.$$
$$\leftarrow p(2,2), p(2,1), p(2,1), 2 \neq 2, 2 \neq 1, 2 \neq 1. \quad \leftarrow p(2,2), p(2,2), p(2,2), 2 \neq 2, 2 \neq 2, 2 \neq 2.$$
$$\leftarrow p(2,2), p(2,3), p(2,3), 2 \neq 2, 2 \neq 3, 2 \neq 3. \quad \leftarrow p(2,3), p(3,1), p(2,1), 2 \neq 3, 3 \neq 1, 2 \neq 1.$$
$$\leftarrow p(2,3), p(3,2), p(2,2), 2 \neq 3, 3 \neq 2, 2 \neq 2. \quad \leftarrow p(2,3), p(3,3), p(2,3), 2 \neq 3, 3 \neq 3, 2 \neq 3.$$
$$\leftarrow p(3,1), p(1,1), p(3,1), 3 \neq 1, 1 \neq 1, 3 \neq 1. \quad \leftarrow p(3,1), p(1,2), p(3,2), 3 \neq 1, 1 \neq 2, 3 \neq 2.$$
$$\leftarrow p(3,1), p(1,3), p(3,3), 3 \neq 1, 1 \neq 3, 3 \neq 3. \quad \leftarrow p(3,2), p(2,1), p(3,1), 3 \neq 2, 2 \neq 1, 3 \neq 1.$$
$$\leftarrow p(3,2), p(2,2), p(3,2), 3 \neq 2, 2 \neq 2, 3 \neq 2. \quad \leftarrow p(3,2), p(2,3), p(3,3), 3 \neq 2, 2 \neq 3, 3 \neq 3.$$
$$\leftarrow p(3,3), p(3,1), p(3,1), 3 \neq 3, 3 \neq 1, 3 \neq 1. \quad \leftarrow p(3,3), p(3,2), p(3,2), 3 \neq 3, 3 \neq 2, 3 \neq 2.$$
$$\leftarrow p(3,3), p(3,3), p(3,3), 3 \neq 3, 3 \neq 3, 3 \neq 3.$$

Therefore, the grounding effort is in $\mathcal{O}(|\mathrm{dom}(\Pi)|^3)$*. In general, the effort yields* $\mathcal{O}(|\mathrm{dom}(\Pi)|^n)$ *where* n *is the largest number of distinct variables in a rule. More recent, intelligent grounders may employ smart procedures towards efficiently producing the ground program (e.g. [GST07], [FLP12]), however, the produced ground program is still potentially of exponential size with respect to the input program.*

Novel Approach on Grounding

To call attention to our novel grounding approach, we recall known Asp complexity results.

For this reason, we identify different program types for logic programs, where the complexity of these types typically ascend with their expressiveness and structure. As seen in Table 1.1, the complexities for *ground* programs are relatively mild: deciding if a *disjunctive* program has an answer set is located at the second level of the polynomial hierarchy [EG95], whereas *normal* (and *tight*) programs yield NP-complete fragments [BF91, MT91].

While variables in *non-ground* programs increase a program's expressiveness and reduce the encoding effort, the complexity reflects the potentially huge costs of grounding, which is why non-ground programs ascend up to NEXPTIME completeness [DEGV01]. However, earlier works (see [EFFW07]), which focused on programs in bounded settings, show that the complexity of programs with bounded predicate arity is more narrow. In particular, the results show that the complexity for non-ground, normal (and tight) programs drop to Σ_2^P completeness and for non-ground, disjunctive programs to Σ_3^P completeness (cf. Table 1.1). What is even more interesting: the exponential blow up in rule instantiation *still holds* for bounded predicate arity despite their milder complexity.

	Ground	Non-Ground (bounded arity)
Tight/Normal Programs	NP-c	Σ_2^P-c
Disjunctive Programs	Σ_2^P-c	Σ_3^P-c
Epistemic Programs	Σ_3^P-c	-

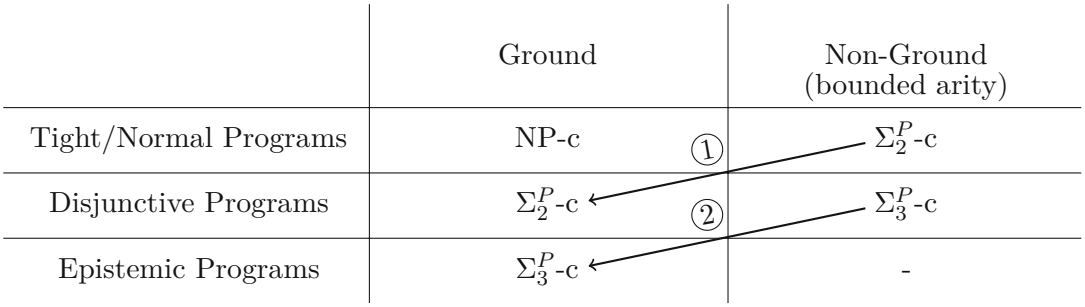

Table 1.1: Known complexity results of the program types discussed in this thesis. The proposed reductions are highlighted with arrows.

Resting upon these results, we can identify that non-ground programs (under bounded arity setting) and ground programs share some of their complexity completeness, an essential property for reducing between problems. In particular, this gives rise to a polynomial reduction between non-ground and ground logic programs (see [Tur39, Pos44, Rog87]), and thus an alternative grounding procedure that delegates certain efforts to the solving process. In the first place, this idea seems qualified for reducing from non-ground, normal (and tight) to ground, disjunctive programs (cf. Fig. 1.1 ①). But as the developement of some competitive epistemic solvers [CFG+20, BMW20, BHW21] suggests, there might even lie potential in a reduction from non-ground, disjunctive Asp programs to ground Elp programs (cf. Fig. 1.1 ②), which are known to be Σ_3^P-complete as well [SE16].

While the reduction itself seems interesting enough, the encoding of such a reduction can be constructed in such a way that the dependencies of predicates in rules bodies are decoupled, i.e. each predicate in the rule's body can be instantiated separately to reduce the number of instantiations. Especially, when assuming large rules bodies with very dense structures, this idea might be particularly useful. The difference of this concept, which we call *body-decoupled grounding*, compared to traditional grounding systems is shown in the following example.

Example 1.3 (Body-Decoupled Grounding)**.** *Recall the program* Π *of Example 1.1.*

$$p(A,B) \vee \bar{p}(A,B) \leftarrow e(A,B) \tag{1.1}$$

$$\leftarrow p(X,Y), p(Y,Z), p(X,Z), X \neq Y, Y \neq Z, X \neq Z. \tag{1.2}$$

As seen above, traditional grounding systems instantiate every variable of a rule at once yielding a grounding effort of $\mathcal{O}(|\mathrm{dom}(\Pi)|^3)$*,* $\mathcal{O}(|\mathrm{dom}(\Pi)|^n)$ *in general where* n *is the largest number of distinct variables in a rule.*

Our approach is different, since it decouples rule bodies during the grounding of Π*. Therefore, body predicates of (1.2) are grounded individually, yielding groundings that are linear in the size of the ground atoms. Here, it corresponds to* $\mathcal{O}(|\mathrm{dom}(\Pi)|^2)$ *due to arity 2.*

In general, the effort yields $\mathcal{O}(|\mathrm{dom}(\Pi)|^c)$ where c is a fixed constant corresponding to the predicate arity.

Main Contributions

By utilizing the aforementioned complexity results we deal with an alternative grounding approach that decouples rule bodies during grounding, which is rectified during solving.

To do so, we try to address grounding from a different perspective and answer the following questions: Can we translate non-ground programs to ground programs without an explicit instantiation of rules, but where instead certain aspects of grounding are delegated to the efficient search procedures of modern ASP (and ELP) solvers? And is this a promising approach, at least, for certain program classes?

Our main contributions are summarized as follows.

1. We present a novel reduction from non-ground, tight logic programs to ground, disjunctive programs (cf. Fig. 1.1 ①) that encodes grounding via search by identifying unsatisfiable ground rules and unjustified (unfounded) atoms. In contrast to traditional grounding, our reduction allows us to *decouple* predicates occurring in the body of a rule, which might be particularly useful for larger bodies with a very dense structure. We show that in general the jump from (non-ground) tight programs to (ground) disjunctive programs cannot be avoided.

2. We extend this approach to non-ground, normal programs, where for ensuring justifiability we additionally encode the idea of orderings (level mappings) in our reduction.

3. As we observe that epistemic solvers are improving, we lift our idea to a reduction to epistemic logic programs. Due to the increased complexity of these programs we can extend this approach to non-ground, disjunctive programs (cf. Fig. 1.1 ②). The presented reduction encodes subset-minimization of answer-sets and can therefore omit foundedness and orderings.

4. We present a prototype that allows to translate critical parts of tight programs using our reduction, thereby empowering the grounding process by decoupling body predicates. Preliminary experiments indicate that this approach can lead to significant speed-ups and a massive reduction in grounding size compared to state-of-the-art grounders.

Publications

At this point, it should be mentioned that this thesis will complement and extend our system NEWGROUND, realizing a program reduction in practice, as well as our

corresponding paper for the *International Joint Conferences on Artificial Intelligence 2022* [BHW22a] (accepted and presented at the conference). Further, this thesis is related to an extended version of [BHW22a] which is currently under review; the translation to epistemic logic programs is entirely new. In addition to the other works, this thesis contains additional examples, more detailed explanations and full proofs.

Further work by the author of this thesis has addressed quantitative reasoning and alternative solving methods for epistemic logic programs [BHW21, BHW22b].

Overview

The rest of this thesis is organized as follows. In the next chapter, Chapter 2, all necessary theoretical background is provided. In particular, this includes introductions to Computational Complexity Theory, Answer-Set Programming and Epistemic Logic Programming. Based on this background information, Chapter 3 first introduces our reduction from non-ground, normal to disjunctive programs and potential optimizations, before lifting the idea to a reduction from non-ground, disjunctive programs to epistemic logic programs. In Chapter 4, NEWGROUND, the implementation of the discussed reduction, will be introduced before analyzing the tool's performance in preliminary experiments. Finally, the last chapter, Chapter 5, discusses and summarizes the findings of this thesis and puts it into perspective of related work. Following that, the outcomes of the results are presented, which give rise to future work.

CHAPTER 2

Background

In this chapter we introduce the preliminary concepts and underlying theory of the approach presented in this work. To this end, Section 2.1 outlines the most relevant concepts of complexity theory, Section 2.2 covers answer-set programming structurally divided into ground and non-ground logic programs, and Section 2.3 introduces the epistemic extension ELP of answer-set programming.

For some definitions, we use mathematical vectors $X{=}\langle x_1, \ldots, x_m\rangle$, $Y{=}\langle y_1, \ldots, y_n\rangle$ in the usual way. We then combine vectors by $\langle X, Y\rangle := \langle x_1, \ldots, x_m, y_1, \ldots, y_n\rangle$ and check whether some x_1 is contained in in a vector X by $x_1 \in X$. Without loss of generality, we assume that elements of vectors are given in any fixed total order; for a given set S, we construct its unique vector by $\langle S\rangle$.

2.1 Computational Complexity

For this section, we assume familiarity with basic complexity theory and will only introduce concepts that are of importance for this thesis. An introduction and more details can be found in [Pap94]. In the following we mainly consider *decision problems* commonly defined as problems that can be posed as a yes–no question of specified sets of inputs.

The problem class P contains all decision problems that can be solved in polynomial time with a deterministic Turing machine. Similar, problems in the problem class NP can be solved in polynomial time using non-deterministic Turing machines. For any class C, there exists a complement class co-C which contains all decision problems whose complement is in C, i.e. a problem $\mathcal{P}$ is in co-C if and only if the complement co-$\mathcal{P}$ of $\mathcal{P}$ is in C. In general, the class P^{C} for any class C includes all decision problems that can be solved in polynomial time with a deterministic Turing machine with access to an oracle

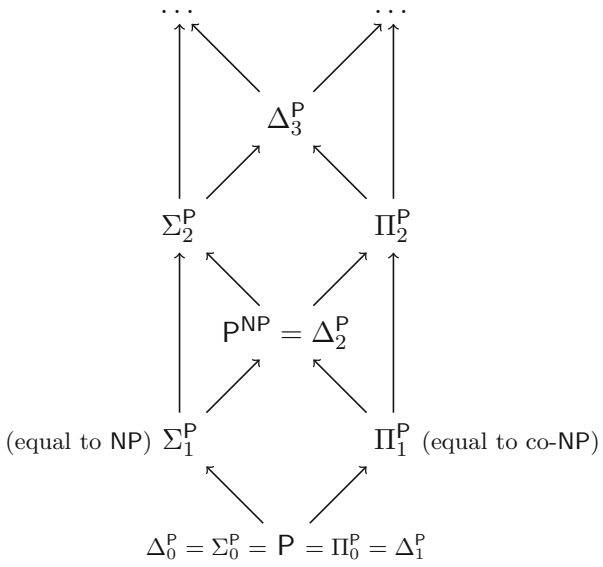

Figure 2.1: Commutative diagram of the polynomial hierarchy.

for the class C, which can be seen as a subroutine able to solve all problems in C running in constant time.

Further, we call a given problem $\mathcal{P}$ C-*complete* if $\mathcal{P}$ is in the complexity class C (membership) and $\mathcal{P}$ is C-*hard*, i.e. any problem of C can be reduced to $\mathcal{P}$ in polynomial time and space (cf. Definition 2.2).

The *polynomial hierarchy* is composed of the three classes Σ_k^{P}, Π_k^{P} and Δ_k^{P}, and can be inductively defined as follows.

$$\Sigma_0^{\mathsf{P}} = \Pi_0^{\mathsf{P}} = \Delta_0^{\mathsf{P}} = \mathsf{P}$$

and for $k \geq 1$*:*

$$\Sigma_{k+1}^{\mathsf{P}} = \mathsf{NP}^{\Sigma_k^{\mathsf{P}}}$$

$$\Pi_{k+1}^{\mathsf{P}} = \textit{co}\text{-}\mathsf{NP}^{\Sigma_k^{\mathsf{P}}}$$

$$\Delta_{k+1}^{\mathsf{P}} = \mathsf{P}^{\Sigma_k^{\mathsf{P}}}$$

Figure 2.1 illustrates the polynomial hierarchy and shows the relationship between the different classes Σ_k^{P}, Π_k^{P} and Δ_k^{P}, where an arrow depicts a class being a subclass, e.g. Σ_1^{P} and Π_1^{P} being a subclass of P^{NP}. One can then speak about the i-th level of the polynomial hierarchy to comprise the classes Σ_i^{P}, Π_i^{P} and $\Delta_{i+1}^{\mathsf{P}}$. Further, the class PSPACE includes all problems that are decidable in polynomial space. It is known that the following chain of inclusions holds

$$\Sigma_0^P \subseteq \Sigma_1^P \subseteq \cdots \subseteq \mathsf{PSPACE},$$

where each is widely believed to be proper.

Earlier works by Wrathall [Wra76] show that *Quantified Boolean Formulas* (QBFs) are especially suitable for showing complexity results, which is why the corresponding decision problem QSAT_i is the showcase problem for determining the complexity of the i-th level of the polynomial hierarchy.

Proposition 2.1. *Given a propositional formula ϕ with its atoms partitioned into $i \geq 1$ pairwise distinct sets $V_1, \ldots, V_i$, deciding whether $\exists V_1, \forall V_2, \ldots, Q_i V_i\ \phi$ is true is Σ_i^{P}-complete, where $Q_i = \exists$ if i is odd and $Q_i = \forall$ if i is even.*

Similar, deciding whether $\forall V_1, \exists V_2, \ldots, Q'_i V_i\ \phi$ is true is Π_i^{P}-complete, W where $Q'_i = \forall$ if i is odd and $Q'_i = \exists$ if i is even.

Many-One Reductions While the idea of problem reductions play an important role in any field where problems have to be solved, it is especially a key technique in complexity theory. As first introduced as a side issue by Turing [Tur39], revisited by Post [Pos44], and later associated with *computability theory* [Rog87], *Turing reductions* pose an important foundation for later scientific work on solving problems by reducing between them. But in particular, *Many-One Reductions*, a stronger form of a Turing reduction (cf. Definition 2.1; Turing reductions allow multiple invocations of the utilized oracle), are an important way of solving problems and determining their complexity due to their properties explained in Definition 2.2 below.

Definition 2.1 (Many-One Reduction)**.** *Given two (decision) problems A, B, let R be a function from instances of problem A to instances B, i.e. each instance a of A is mapped to an instance $b = R(a)$ of B. When solving the instance $R(a)$ with the algorithm for problem B, the answer is already the correct answer to the instance a of A, we say a and $R(a)$ are equivalent, i.e. a is a positive instance of $A \Leftrightarrow R(a)$ is a positive instance of B.*

Notice that, many-one reductions are typically used under resource restrictions. *Polynomial-Time Reductions*, which are generally considered, transform instances of one decision problem into instances of another decision problem in polynomial time. This prevents hiding the complexity of a problem in the reduction and makes the complexity comparison possible. Therefore, we only focus on polynomial-time many-one reductions and denote a reduction with $\mathcal{P} \leq_R \mathcal{P}'$ if problem $\mathcal{P}$ can be reduced to $\mathcal{P}'$.

Definition 2.2 (Hardness & Completeness)**.** *Let C be a complexity class and let $\mathcal{P}$ be a problem. $\mathcal{P}$ is called C-hard if any problem $\mathcal{P}' \in \mathsf{C}$ is reducible to $\mathcal{P}$. $\mathcal{P}$ is called C-complete if $\mathcal{P}$ lies in C and $\mathcal{P}$ is C-hard.*

These properties can be used for different use cases. On the one hand, we can use reductions to prove the complexity (hardness) of a problem. We can argue that if a problem $\mathcal{P}$ is known to be C-hard and a problem $\mathcal{P}$' can be reduced to $\mathcal{P}$, i.e. $\mathcal{P}' \leq_R \mathcal{P}$, then $\mathcal{P}'$ is also C-hard. On the other hand, reductions can be used for solving problems

by reducing them to a problem for which a sophisticated, superior solver exists, e.g. many NP problems are reduced to Sat because of its powerful solvers.

2.2 Answer-Set-Programming

Answer-Set Programming (often abbreviated to ASP) deals with so-called logic programs, which is why it also often referred to as logic programming under the stable-model semantics, and furthermore, with finding the associated solutions, called *Answer-Sets*, of those. Since this thesis addresses methods for grounding, the process needed to solve non-ground logic programs, a clear distinction between ground and non-ground programs is made, which is why they are introduced separately. The following definitions are based on [BET11, JN16].

2.2.1 Ground Answer-Set Programming

Let ℓ, m, n be non-negative integers such that $\ell \leq m \leq n$; a_1, ..., a_n be distinct propositional atoms. We refer to a *propositional atom* or the negation of it by *literal*. A ground *(disjunctive) program* P is a set of *(disjunctive)* rules of the form

$$a_1 \vee \ldots \vee a_\ell \leftarrow a_{\ell+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n. \tag{2.1}$$

For a rule r, we let $H_r := \{a_1, \ldots, a_\ell\}$ be the set of atoms appearing in the head of a rule r, $B_r^+ := \{a_{\ell+1}, \ldots, a_m\}$ be the set of atoms appearing the body of the rule without negation, and $B_r^- := \{a_{m+1}, \ldots, a_n\}$ be the set of atoms appearing in the body with negation. We denote the sets of *atoms* occurring in a rule r or in a program P by $\mathrm{at}(r) := H_r \cup B_r^+ \cup B_r^-$ and $\mathrm{at}(P) := \bigcup_{r \in P} \mathrm{at}(r)$.

Example 2.1. *Consider the program*

$$P_1 := \{\overbrace{a \vee b \leftarrow}^{r_1}; \overbrace{c \vee d \leftarrow e}^{r_2}; \overbrace{e \leftarrow b, \neg d}^{r_3}; \overbrace{e \leftarrow \neg b}^{r_4}; \overbrace{d \leftarrow b, \neg e}^{r_5}; \overbrace{b \leftarrow d, \neg e}^{r_6}\}.$$

As defined above, we can identify the following sets of atoms appearing in the rules.

$$\begin{aligned}
H_{r_1} &= \{a, b\}, B_{r_1}^+ = \{\}, B_{r_1}^- = \{\} \\
H_{r_2} &= \{c, d\}, B_{r_2}^+ = \{e\}, B_{r_2}^- = \{\} \\
H_{r_3} &= \{e\}, B_{r_3}^+ = \{b\}, B_{r_3}^- = \{d\} \\
H_{r_4} &= \{e\}, B_{r_4}^+ = \{\}, B_{r_4}^- = \{b\} \\
H_{r_5} &= \{d\}, B_{r_5}^+ = \{b\}, B_{r_5}^- = \{e\} \\
H_{r_6} &= \{b\}, B_{r_6}^+ = \{d\}, B_{r_6}^- = \{e\}
\end{aligned}$$

Accordingly, for the set of atoms appearing in the program we have $\mathrm{at}(P) = \{a, b, c, d, e\}$.

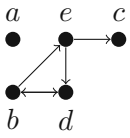

Figure 2.2: Dependency graph D_{P_1} of P_1 (cf. Example 2.2).

A rule r is *normal* if $|H_r| \leq 1$ and a program Π is *normal* if all its rules are normal, otherwise the program is called *disjunctive*. Further, we can identify *tight* programs by the following definition.

Definition 2.3 (Dependency Graph)**.** *The* dependency graph $\mathcal{D}_P$ *of a ground program* P *is the directed graph defined on the set* $\bigcup_{r\in\Pi} H_r \cup B_r^+$ *of atoms, where for every rule* $r \in P$ *two atoms* $a \in B_r^+$ *and* $b \in H_r$ *are joined by an edge* (a, b)*. A program* P *is called* tight *if* $\mathcal{D}_\Pi$ *has no directed cycle [Fag94].*

Example 2.2. *Recall the program* P_1 *of Example 2.1.*

We can observe that the program Π_1 *is not tight, since the dependency graph* D_{P_1}*, seen in Figure 2.2, contains the directed cycle* $\{b, d, e\}$*. Further, since we have rules* r_1 *and* r_2*, where we have* $|H_r| > 1$ *for* $r \in \{r_1, r_2\}$*, the program is not normal, but disjunctive.*

Definition 2.4 (Interpretation)**.** *An interpretation* I *is a set of atoms. A rule* r *is satisfied by an interpretation* I *if* $(H_r \cup B_r^-) \cap I \neq \emptyset$ *or* $B_r^+ \setminus I \neq \emptyset$*. An interpretation* I *is a model of a program* P *if it satisfies every rule* $r \in P$*, in symbols* $I \models P$*. The Gelfond-Lifschitz (GL) reduct of* P *under* I *is the program* P^I *obtained from* P *by first removing all rules* r *with* $B_r^- \cap I \neq \emptyset$ *and then removing all* $\neg z$ *where* $z \in B_r^-$ *from the remaining rules* r *[GL91]. An interpretation* I *is an Answer-Set of a program* P*, if* I *is a minimal model (w.r.t.* $\subseteq$*) of* P^I*, i.e.* I *satisfies every rule* r *of the GL reduct* P^I *and every proper subset of* I *does not satisfy* P^I*. We refer to the set of answer sets of a given program* P *by* $AS(P)$*.*

Example 2.3. *Recall the program* P_1 *of Example 2.1.*

We can show that the set $\{a, c, e\}$ *is an answer set of* P_1*, since it is a minimal model of the GL reduct* $P_1^{\{a,c,e\}} = \{a \vee b \leftarrow,\ c \vee d \leftarrow e,\ e \leftarrow b,\ e \leftarrow\}$*. Furthermore, we can prove the set* $\{a, c, e\}$ *as an answer, since it is a minimal model of* $P_1^{\{b,d\}} = \{a \vee b \leftarrow,\ c \vee d \leftarrow e,\ d \leftarrow b,\ b \leftarrow d\}$*. In similar fashion, one can find all answer sets of the program, which are* $AS(P_1) = \{\{a, c, e\}, \{a, d, e\}, \{b, c, e\}, \{b, d\}\}$*.*

Proposition 2.2. *The decision problem of deciding whether an* ASP *program has an answer set, which is referred to as* consistency*, is (in general)* Σ_2^P*-complete [EG95]. If the input is restricted to normal or tight programs, the complexity drops to* NP*-completeness [BF91, MT91], as seen in Table 2.1.*

Decision Problem	Fragment	Ground	Non-Ground (bounded arity)
Tight Asp	tight programs	NP [MT91]	Σ_2^P [EFFW07]
Normal Asp	normal programs	NP [BF91]	Σ_2^P [EFFW07]
Disjunctive Asp	disjunctive programs	Σ_2^P [EG95]	Σ_3^P [EFFW07]
Elp	epistemic programs	Σ_3^P [SE16]	-

Table 2.1: Decision problems related to the answer-set programming formalism that are discussed in the course of this work. The column "Fragment" states the corresponding fragment of logic programs, while the columns "Ground" and "Non-Ground (bounded arity)" list their respective complexity completeness.

The following characterization of answer sets is often applied for normal programs [LZ03, Jan06].

Definition 2.5 (Orderings)**.** *Let I be a model of a normal program P and φ be a function (ordering) $\varphi : I \to \{0, \ldots, |I| - 1\}$ over I. We say a rule $r \in P$ is suitable for justifying $a \in I$ if (i) $a \in H_r$, (ii) $B_r^+ \subseteq I$, (iii) $I \cap B_r^- = \emptyset$, as well as (iv) $I \cap (H_r \setminus \{a\}]) = \emptyset$. An atom $a \in I$ is founded if there is a rule $r \in P$ justifying a, which is the case if r is suitable for justifying a and $\varphi(b) < \varphi(a)$ for every $b \in B_r^+$, and unfounded otherwise. Then, I is an answer set of P if (i) I is a model of P, and (ii) I is founded, i.e., every $a \in I$ is founded. For tight programs, the ordering φ is not needed.*

Example 2.4. *Recall the program P_1 of Example 2.1.*

As shown in Example 2.3, the set $I := \{a, c, e\}$ is an answer set of P_1. This can also be proven by $I \models P_1$ and a level mapping $\varphi := \{a \mapsto 0, e \mapsto 1, c \mapsto 2\}$. Then, the atom a is founded by rule r_1, atom e by rule r_4 and atom c by rule r_2.

Established ASP Techniques Through years of research in logic programming several techniques have established and became so to say state-of-the-art.

The encoding of problems into ASP has always been an crucial step, especially since it is can increase the solution finding for a given program tremendously. Today, programs are typically designed following the established Guess-and-Check paradigm [EP06]. This way programs are written in such a way that solution candidates are non-deterministically guessed first, before eliminating invalid ones by enforcing constraints that each solution must comply (cf. Example 1.1).

As already mentioned in the Introduction, Gebser et al. maintain a leading collection of answer set solving tools in form of their Potassco system, with the main driver clingo [GKK+11, GKKS19]. While, of course, the tools support *grounding* (see Section 2.2.2), there are also more advanced expressions and classical negation that are supported. However, during preprocessing and grounding these are usually reduced

to standard logic programs and therefore present user-friendly adaptions for encoding problems. Especially interesting, since it is later used for optimizing the encodings, is the idea of *aggregates* of the form

$$s_1 \prec_1 \alpha \{ t1 : L1; \ldots; t_n : L_n \} \prec_2 s_2$$

where t_i and L_i are tuples of terms (used for weighting/prioritizing) and literals, α the name of the function (`#count`, `#sum`, `#sum+`, `#min`, or `#max`) and $\prec_1$, $\prec_2$ comparison predicates to the terms s_1, s_2. Naturally, this extension to the original ASP syntax allows the forming of values from groups of selected items and expressing conditions over them, e.g. a minimum count of conditions being evaluated to true. Further details and syntactic additions of the CLINGO system can be found in [GKKS19, CDF+20].[1]

Another common practice is the *saturation technique* [EG95, EGM97].While saturation is a quite error-prone and sophisticated encoding technique, it allows the exploiting of ASPs full expressive power (Σ_2^{P}) by encoding problems in disjunctive ASP while complementing the above-mentioned Guess-and-Check paradigm [EP06]. Saturation is typically constructed by encoding the co-NP-check in the following steps, as described in [ABC+15]. First of all, we guess a solution candidate S for which we want to know if it withstands every possibility for not being a valid solution. To do so, we also guess (using disjunction) a counter candidate C as a potential witness for S being invalid. If C is not such a witness, we derive a designated atom A that causes all atoms of the (second) guess to be set to true ("saturation"). By doing so, all guesses of potential witnesses collapse to a unique maximal answer set if a valid solution has been guessed. Now, we either have guessed a witness C for which we can kill the solution candidate S using a constraint, or we have an invalid solution candidate S which is discarded by the minimal model semantics because each model that does not result in a witness is saturated.

Example 2.5. *Assume the following simple QBF* $\psi := \exists a \forall b (a \vee \neg b)$. *Recall that the problem of deciding whether QBFs of the form* $\exists V_1 \forall V_2 \phi$ *are true is* Σ_2^{P}*-complete (cf. Definition 2.1). Listing 2.1 illustrates how to solve this problem using the disjunctive program* P_2. *As explained above, the program uses saturation and models the variables* a, b *of* ψ *with the atoms* ta/fa *and* tb/fb, *intuitively as true/false. Notice that* ψ *has two satisfying models with* $M_1(a) = M_2(a) = true$, $M_1(b) = false$ *and* $M_2(b) = true$. *In comparison, when assuming program* P_2 *with a guessed to true, i.e.* $M(ta) = true$ *and* $M(fa) = false$, *we consequently have* $M(sat) = true$ *due to Line 6,* $M(tb) = true$ *due to Line 10 and* $M(fb) = true$ *due to Line 11. Observe that a program* P_2^{rel}, *a relaxed scenario of* P_2 *without Line 14 to allow potentially unsatisfied solutions, cannot have a* $M' \subseteq M$ *and* $M' \neq M$, *such that* $M' \models P_2^{rel}$. *Since we require* $M' \subseteq M$, $M'(fa) = true$ *cannot hold, i.e. we get that* $M'(ta) = true$ *from Line 2. In consequence, we get* $M(sat) = true$, $M(tb) = true$ *and* $M(fb) = true$ *- leaving us with* $M' = M$, *proving* M *as an answer set of* P_2. *Observe that by similar reasons there cannot exist any model* M'' *of* P_2, *which is*

[1] More information about the clingo input language can be found in the Potassco guide at `https://potassco.org/doc/`.

exactly as M but additionally with $M''(fa) = true$, since then $M'' \supset M$. To show that M is the only answer set of P_2, assume another model M^* of P_2 with $M^*(fa) = true$. To satisfy Line 14 and $M^* \models P_2$, we have to set $M^*(fb) = true$. However, there exists a model $M^{**} \subset M^*$ of P_2^{rel}, where $M^{**}(tb) = true$ and $M^{**}(sat) = false$, invalidating M^* as potential answer set.

```
1  % Guess truth values of variables a and b
2  ta ∨ fa.
3  tb ∨ fb.

5  % Model the cases where ψ evaluates to true
6  sat ← ta.
7  sat ← fb.

9  % Saturize over the ∀-quantified variables, if ψ evaluates to true
10 tb ← sat.
11 fb ← sat.

13 % Ensure satisfiability
14 ← not sat.
```

Listing 2.1: Program P_2 using the saturation technique to solve ψ of Example 2.5.

2.2.2 Non-Ground Answer-Set Programming

As said, we clearly differentiate between ground and non-ground ASP programs. Compared to the former, the latter introduces the use of variables to address all possible atoms. While this allows for shorter and more versatile problem encodings, therefore enriching ASPs expressiveness, the complexity for solving these suffer. Similar to ground Asp, we define non-ground Asp as follows.

Let $p_1, \dots p_n$ be predicates, where each takes *arity* $|p_i|$ many variables for $1 \leq i \leq n$. A *non-ground program* Π is a set of *non-ground rules* of the form

$$p_1(X_1) \vee \dots \vee p_\ell(X_\ell) \leftarrow p_{\ell+1}(X_{\ell+1}), \dots, p_m(X_m), \neg p_{m+1}(X_{m+1}), \dots, \neg p_n(X_n). \tag{2.2}$$

where for every *variable vector* X_i we have $|X_i| = |p_i|$. Whenever we have $x \in \langle X_1, \dots, X_\ell, X_{m+1}, \dots, X_n \rangle$, then $x \in \langle X_{\ell+1}, \dots, X_m \rangle$, such that rules can be considered *safe*, i.e. all variables that are used in a rule appear in some positive literal in the body. For a non-ground rule r, we let similar to ground rules $H_r := \{p_1(X_1), \dots, p_\ell(X_\ell)\}$, $B_r^+ := \{p_{\ell+1}(X_{\ell+1}), \dots, p_m(X_m)\}$, $B_r^- := \{p_{m+1}(X_{m+1}), \dots, p_n(X_n)\}$, and $\text{var}(r) := \{x \in X \mid p(X) \in H_r \cup B_r^+ \cup B_r^-\}$. Further, we use $\text{heads}(\Pi) := \{p(X) \in H_r \mid r \in \Pi\}$ to refer to the predicates (including the variable vector), and $\text{hpreds}(\Pi) := \{p \mid p(X) \in \text{heads}(\Pi)\}$ for the predicate names only, occurring in the head of a rule. Without loss of generality, we assume that *variables*

are unique per rule, i.e., for every two rules $r, r' \in \Pi$, we have $\text{var}(r) \cap \text{var}(r') = \emptyset$. The rule size corresponds to $\|r\| := |B_r^+| + |B_r^-| + |H_r|$ and program size $\|\Pi\| := \sum_{r \in \Pi} \|r\|$. The attributes *disjunctive* and *normal*, as earlier defined for ground programs in Section 2.2.1, naturally carry over to non-ground rules and programs. Non-ground, *tight* programs can be defined as follows.

Definition 2.6 (Non-ground Dependency Graph)**.** *The* dependency graph $\mathcal{D}_\Pi$ *of a non-ground program* Π *is the directed graph defined on the set* $\text{hpreds}(\Pi)$ *of head-predicates of the rules* $r \in \Pi$*. There is a directed edge from* p *to* q *whenever there is a rule* $r \in \Pi$ *with* $p(X) \in B_r^+$ *and* $q(Y) \in H_r$*.*

A non-ground program Π *is called* tight *if* $\mathcal{D}_\Pi$ *has no directed cycle.*

In order to solve a given non-ground program Π, the *grounding* of the program, the process of instantiating the non-ground rules, is required. For the process of grounding Π, we require a given set $\mathcal{F}$ of atoms, reflecting the *facts*, i.e., atoms of the form $p(D)$ with p being a predicate of Π and D being a vector over *domain values* of size $|D| = |p|$. We say that D is part of the *domain* of Π (with respect to $\mathcal{F}$), defined by $\text{dom}(\Pi) := \{d \in D \mid p(D) \in \mathcal{F}\}$. We refer to the *domain vectors* over $\text{dom}(\Pi)$ for a variable vector X of size $|X|$ by $\text{dom}(X)$. Let D be a domain vector over variable vector X and vector Y contain only variables of X. We refer to the *domain vector of* D *restricted to* Y by D_Y.

Definition 2.7 (Grounding)**.** *The* grounding $\mathcal{G}(\Pi)$ *consists of* $\mathcal{F}$ *and ground rules obtained by replacing each rule* r *of Form (2.2) for every domain vector* $D \in \text{dom}(\langle \text{var}(r) \rangle)$ *by*

$$p_1(D_{X_1}) \vee \ldots \vee p_\ell(D_{X_\ell}) \leftarrow p_{\ell+1}(D_{X_{\ell+1}}), \ldots, p_m(D_{X_m}), \\ \neg p_{m+1}(D_{X_{m+1}}), \ldots, \neg p_n(D_{X_n}).$$

The effort of grounding a program Π *is therefore in* $\mathcal{O}(|\text{dom}(\Pi)|^n)$*, where* n *is the maximum number of distinct variables over all rules.*

Interestingly, for fixed arity the complexity of deciding consistency of a non-ground program increases only by one level.

Proposition 2.3 (Grounding Complexity for fixed arity [EFFW07])**.** *Let* Π *be any tight (normal) or disjunctive, non-ground program, where every predicate has arity at most* a*. Then, deciding whether* $\mathcal{G}(\Pi)$ *admits an answer set is complete for* Σ_2^P *or* Σ_3^P*, respectively.*

We can not note that, unless using growing predicate arities, with non-ground Asp we can not encode problems above the polynomial hierarchy, e.g., as commonly believed, PSPACE-complete problems. Further, these complexity results give rise to an avoidance of the grounding of non-ground programs with bounded predicate arity by polynomially mapping/reducing to ground programs. [EFFW07]

Example 2.6. *Consider the tight, non-ground program* $\Pi := \{r\}$ *with* $r = a(X,Y) \leftarrow b(X), c(Y,Z)$ *and* $\mathcal{F} := \{b(1).\ c(1,2).\}$. *Observe that, while* $\mathrm{dom}(\Pi) = \{1,2\}$ *we have* $\mathrm{dom}(X) = \{1\}$ *and* $\mathrm{dom}(Y) = \mathrm{dom}(Z) = \{1,2\}$. *The grounding* $P = \mathcal{G}(\Pi)$ *of* Π *consists of:*

$$\{a(1,1) \leftarrow b(1), c(1,1).\ a(1,1) \leftarrow b(1), c(1,2).\\ a(1,2) \leftarrow b(1), c(2,1).\ a(1,2) \leftarrow b(1), c(2,2).\}$$

The only answer set of P *is* $\{b(1), c(1,2), a(1,1)\}$.

Another frequently considered example, which can be elegantly encoded using non-ground Asp and the Guess-and-Check paradigm (cf. Section 2.2.1), is the 3-colorability of a given graph, i.e. if a given graph is *3-colorable*.

Example 2.7. *Listing 2.2 shows the encoding for checking if a graph, given by set of facts* $\mathcal{F}$*, is 3-colorable. Observe that, while the non-ground program is only composed of two rules, the grounding yields an exponentially larger program.*

To exemplify, considering a set $\mathcal{F}$ *of facts, such that* $|\mathrm{dom}(X)| = |\mathrm{dom}(Y)| = n$ *with* n *being a large number. Then the grounding of Line 5 in Listing 2.2 yields the* n^2 *for each color, i.e. multiplied by three, by the means of combinatorics.*

```
1 % Guess one of the three colors for each node of the graph
2 col(red, N) ∨ col(green, N) ∨ col(blue, N) :- node(N).

4 % Check whether no adjacent nodes have the same color
5 :- edge(X,Y), col(C,X), col(C,Y).
```

Listing 2.2: Encoding for the 3-Colorability of a given graph.

2.3 Epistemic Logic Programming

An *epistemic logic program (ELP)* P is an extension of a logic program as defined above. The following definitions are based on [Gel91, Tru11, KWB+15, SE16].

In addition to standard logic program, where each rule body can contain *epistemic literals*. An epistemic literal is a formula of the form **not**ℓ, where ℓ is a literal in the classical sense and **not** is the *epistemic negation* operator.

Let k, m, j, n be non-negative integers such that $k \leq m \leq j \leq n$ and $a_1, \ldots, a_n$ be distinct propositional atoms. An *epistemic logic program (ELP)* is a set Π of *ELP rules* of the form

$$a_1 \vee \cdots \vee a_k \leftarrow \ell_{k+1}, \ldots, \ell_m, \xi_{m+1}, \ldots, \xi_j, \neg\xi_{j+1}, \ldots, \neg\xi_n. \tag{2.3}$$

where each ℓ_i is a literal over atom a_i, and each ξ_i is an epistemic literal of the form **not**ℓ_i, where ℓ_i is a literal over atom a_i. Similarly to logic programs, let $H_r = \{a_1, \ldots, a_k\}$, and

let $B_r = \{\ell_1, \ldots, \ell_m, \xi_1, \ldots, \xi_j, \neg\xi_{j+1}, \ldots, \neg\xi_n\}$ that is, the set of elements appearing in the rule body.

Then, $\mathsf{at}(r) := \{a_1, \ldots, a_n\}$ denotes the set of atoms occurring in an ELP rule r, $\mathsf{e\text{-}at}(r) := \{a_{m+1}, \ldots, a_n\}$ denotes the set of *epistemic atoms*, i.e., those used in epistemic literals of r, and $\mathrm{var}(r) := \mathsf{at}(r) \setminus \mathsf{e\text{-}at}(r)$ refers to the *non-epistemic* atoms of r. These notions naturally extend to programs. In a rule we sometimes write $\mathbf{K}\ell$ and $\mathbf{M}\ell$ for a literal ℓ, which refers to the expressions $\neg\mathbf{not}\ell$ and $\mathbf{not}\neg\ell$, respectively.

Definition 2.8 (World View Interpretation)**.** *Given an ELP* P*, a* world view interpretation (WVI) *I for P is a consistent set I of literals over a set $A \subseteq \mathsf{at}(\mathsf{P})$ of atoms, i.e., $I \subseteq \{a, \neg a \mid a \in A\}$ such that there is no $a \in A$ with $\{a, \neg a\} \subseteq I$.*

Intuitively, for a WVI I every $\ell \in I$ is considered as "known" and every $a \in A$ with $\{a, \neg a\} \cap I = \emptyset$ is treated as "possible". We denote the WVI over a set $X \subseteq \mathsf{at}(\mathsf{P})$ of atoms obtained by restricting I to $Y = (A \cap X)$ by $I_{|X} := I \cap \{a, \neg a \mid a \in Y\}$. Next, we define compatibility with a set of interpretations.

Definition 2.9 (WVI Compatibility)**.** *Let $\mathcal{I}$ be a set of interpretations over a set A of atoms. Then, a WVI I is* compatible *with $\mathcal{I}$ if:*

1. *$\mathcal{I} \neq \emptyset$;*
2. *for each atom $a \in I$, it holds that for each $J \in \mathcal{I}$, $a \in J$;*
3. *for each $\neg a \in I$, we have for each $J \in \mathcal{I}$, $a \notin J$;*
4. *for each atom $a \in A$ with $\{a, \neg a\} \cap I = \emptyset$, there are $J, J' \in \mathcal{I}$, such that $a \in J$, but $a \notin J'$.*

While there are many semantics for ELPs [Gel91, Tru11, KWB+15, SE16], we will use the approach by Morak [Mor19] in order to define the semantics of an ELP. His approach [Mor19] follows the semantics defined in [SE16], but uses a different formal representation. Note that, however, our results can be adapted to other "reduct-based" semantics, by changing the definition of the reduct appropriately.

Definition 2.10 (Epistemic Reduct)**.** *The epistemic reduct [Gel91] of program P w.r.t. a WVI I over A, denoted P^I, is defined as $\mathsf{P}^I = \{r^I \mid r \in \Pi\}$ where r^I denotes rule r where each epistemic literal $\mathbf{not}\ell$, whose atom is also in A, is replaced by $\bot$ if $\ell \in I$, and by $\top$ otherwise. Note that P^I is a plain logic program with all occurrences of epistemic negation removed.*

Then, a WVI I over $\mathsf{at}(\mathsf{P})$ is a *world view (WV)* of P iff I is compatible with the set $AS(\mathsf{P}^I)$. The set of WVs of an ELP P is denoted $WVS(\mathsf{P})$.

Of course, as an ASP extension, ELPs can also be encoded in a non-ground form, which requires grounding before the solving of a program. This thesis will only cover ground

ELPs, which is why we restrict the background to this category. The problem of deciding the existence of a world view for a ground ELP is known to be Σ_3^P-complete [Tru11], as seen in Table 2.1.

Example 2.8. *Consider the ground ELP program*

$$\mathsf{P} := \{a, b;\ g \leftarrow \neg \mathbf{K} a;\ a \leftarrow \neg \mathbf{K} g\}$$

When constructing a WVI I over $\mathsf{e\text{-}at}(\Pi)$ one guesses for each atom $a \in \mathsf{e\text{-}at}(\Pi)$ either (1) $a \in I$, (2) $\neg a \in I$ or (3) $\{a, \neg a\} \cap I = \emptyset$ as described earlier, i.e., for the two atoms in $\mathsf{e\text{-}at}(\Pi)$ we obtain 3^2 possibilities. Each WVI I can be checked with the corresponding epistemic reduct P^I by verifying Definition 2.9 for $AS(\mathsf{P}^I)$.

Consider $I_1 = \{a, \neg g\}$ with its epistemic reduct $\Pi^{I_1} := \{a, b;\ g \leftarrow \bot;\ a\}$. Note that the epistemic reduct is indeed a plain logic program by the semantics of logic programs and rules r with $\bot \in B_r^+$ or $\top \in B_r^-$ can obviously be dropped. Since $AS(\mathsf{P}^{I_1}) = \{\{a\}\}$, compatibility of I_1 can be checked trivially which validates I_1 as WV of P (which is the only one).

CHAPTER 3

Grounding Approach via Reduction

This chapter introduces a novel grounding approach we call *Body-Decoupled Grounding*, which combines the concept of problem reductions and the known complexities of non-ground programs to reduce between non-ground and ground logic programs. This reduction is encoded in such a way that body atoms are decoupled, minimizing the combinatorial effort of instantiating variables.

In the following we motivate this idea with an example before Section 3.1 introduces our approach for tight logic programs, which is extended to normal programs in Section 3.2. In Section 3.3 the idea is lifted to disjunctive programs, where a reduction and the necessary changes to reduce more complex program types are presented.

Motivation As implied in Chapter 2, the technique of reducing problems based on complexity results allow the idea of reducing a given type of non-ground logic program into a ground program of the same complexity level. While the reduction is interesting itself, we aim for a performance advantage over traditional grounding systems with our encoding.

Traditional grounding systems are known to be inefficient for programs or rules, which are composed of long rule bodies and higher number of variables. This is due to the common grounding-procedure of generating ground rules from non-ground rules by simply instantiating the occurring variables with all possible domain value combinations. This instantiation results in an exponential grounding size of the program, which, interestingly, holds for programs with bounded predicate arity as well (cf. Table 2.1). In literature this problem is usually referred to as *ASP grounding bottleneck*.

In comparison, with our approach we introduce a concept named *body-decoupled grounding*. This type of grounding utilizes the idea of reducing between non-ground program types

by using a translating encoding that reduces the combinatorial effort (and therefore grounding size) by decoupling dependencies between different predicates of rule bodies, i.e each predicate in the rule's body can be instantiated separately to minimize the number of instantiations. Further, with this approach we intuitively shift parts of the complexity into the solving (cf. ground-and-solve systems), which is why we also call our approach *Body-Decoupled Grounding via Solving*.

As already appearing in Chapter 1, the potential of this idea can be shown with the following Example.

Example 3.1. *Assume the following non-ground program* Π *that decides in (3.1) for each edge (e) of a given graph, whether to pick it (p) or not* ($\bar{p}$)*. Then, in (3.2) it is ensured that the choice of edges does not form triangles.*

$$p(A,B) \vee \bar{p}(A,B) \leftarrow e(A,B) \tag{3.1}$$

$$\leftarrow p(X,Y), p(Y,Z), p(X,Z), X \neq Y, Y \neq Z, X \neq Z. \tag{3.2}$$

The typical grounding effort of (3.2) is in $\mathcal{O}(|\mathrm{dom}(\Pi')|^3)$*. Our approach grounds body predicates of (3.2) individually, yielding linear bounds in the size of the ground atoms, i.e.,* $\mathcal{O}(|\mathrm{dom}(\Pi')|^2)$.

In the following we will first present our approach for tight and normal programs. To ensure traceability, we start with the procedure for tight programs followed by lifting the idea to normal programs, which is in close succession and can be seen as an extension. Only then, the reduction for disjunctive programs is presented, which, while it is based on the former procedure, requires additional effort due to the program's complexity.

3.1 Body-Decoupled Grounding for Tight ASP

To ensure a comprehensible entry into our reduction, we first present our approach for cycle-free programs without disjunction. To this end, we assume a given non-ground, tight program Π and a set $\mathcal{F}$ of facts. Then, our reduction relies on the following sets of variables. For each predicate $p(X)$ in heads(Π), we use every instantiation of $p(X)$ and its negation $\bar{p}(X)$ over dom(Π), resulting in atoms AtPred $:= \{p(D), \bar{p}(D) \mid p(X) \in \mathrm{heads}(\Pi), D \in \mathrm{dom}(X)\}$, which are used as a guess for (potential) answer sets. Note that the atoms over $\bar{p}(X)$ are for technical reasons only, and are not needed when employing e.g., choice rules [SNS02].

In addition to AtPred and in accordance with the semantics of Asp, we require to ensure (i) satisfiability and (ii) foundedness. For (i) computing models of rules, we require atoms AtSat $:= \{\mathrm{sat}, \mathrm{sat}_r, \mathrm{sat}_x(d) \mid r \in \Pi, x \in \mathrm{var}(r), d \in \mathrm{dom}(x)\}$, where sat_r indicates the satisfiability of a non-ground rule r and sat the satisfiablity of the program, respectively. Atoms of the form $\mathrm{sat}_x(d)$ are used to indicate that we assign variable x of non-ground rule r to domain value $d \in \mathrm{dom}(x)$ when checking for satisfiability.

For (ii) ensuring foundedness, we use variables AtUf $:= \{\mathrm{uf}_r(D_X), \mathrm{uf}_y(D_{\langle X,y\rangle}) \mid r \in \Pi, D \in \mathrm{dom}(\langle \mathrm{var}(r)\rangle), h(X) \in H_r, y \in \mathrm{var}(r), y \notin X\}$. Intuitively, $\mathrm{uf}_r(D)$ indicates that r

fails to justify $h(D)$ for head predicate $h(X) \in H_r$ and domain vector $D \in \mathrm{dom}(h)$, where $\mathrm{uf}_y(D_{\langle X,y \rangle})$ refers to the assigned domain value d that variable y gets in this foundedness check for $h(D)$, i.e. we ensure foundedness with atoms deriving unfoundedness. Overall, the number of atoms used for our approach is limited by the largest predicate arity as follows.

Observation 3.1 (Auxiliary Atoms)**.** *The number of atoms in* $|\mathrm{AtPred} \cup \mathrm{AtSat} \cup \mathrm{AtUf}|$ *is bounded by* $\mathcal{O}(\|\Pi\| \cdot |\mathrm{dom}(\Pi)|^{a+1})$*, where* a *is the largest predicate arity.*

After the declaration of the used atoms, we are in position to explain our reduction for tight programs. The overall idea consists of three parts and is part of the reduction $\mathcal{R}$, which transforms the non-ground, tight program Π into a ground, disjunctive program $\mathcal{R}(\Pi)$ consisting of facts $\mathcal{F}$ and the rules given in Figure 3.1. Rules ($\mathcal{R}1$) take care of guessing answer set candidates, then Rules ($\mathcal{R}2$)–($\mathcal{R}7$) ensure (i) satisfiability, and Rules ($\mathcal{R}8$)–($\mathcal{R}11$) model (ii) foundedness.

Interestingly, the only disjunction that is indeed crucial and cannot be modeled via choice rules, is the disjunction part of Rules ($\mathcal{R}2$) for (i) satisfiability, which is responsible for guessing and saturating assignments of variables to domain values. The construction is such that whenever for a non-ground rule $r \in \Pi$, there is an assignment of variables to domain values such that the resulting ground rule is satisfied, Rules ($\mathcal{R}3$) or ($\mathcal{R}4$) yield sat_r. If such an atom sat_r can be derived for all non-ground rules of $r \in \Pi$, we follow sat by Rules ($\mathcal{R}5$), which is mandatory (cf. Rules ($\mathcal{R}7$)). Then, Rules ($\mathcal{R}6$) apply *saturation*, which causes the assignment of all domain values to every variable. Assuming that a grounding of a rule $r \in \Pi$ was not satisfied, then there would exist a $\subseteq$-smaller model of the reduct, invalidating the answer set candidate. Intuitively, the construction takes care that there is an answer set of $\mathcal{R}(\Pi)$, only if the grounding of Π admits an answer set.

Rules ($\mathcal{R}8$) are for (ii) preventing unfoundedness, ensuring that for each head atom $h(D)$ contained in a model of $\mathcal{R}(\Pi)$, variables get assigned domain values for proving foundedness. Unjustifiability of a rule $r \in \Pi$ for atom $h(D)$ is derived by Rules ($\mathcal{R}9$) and ($\mathcal{R}10$), which is prevented by Rules ($\mathcal{R}11$).

Example 3.2. *Consider the non-ground, tight program* $\Pi{=}\{a(X,Y) \leftarrow b(X), c(Y,Z)\}$ *and facts* $\mathcal{F} = \{b(1).\ c(1,2).\}$ *from Example 2.6. Grounding the program as described above and shown in Figure 3.1, results in the ground program* $P = \mathcal{R}(\Pi)$ *of Figure 3.2. The answer sets of* P *restricted to symbols* a, b, c *of* Π *yield* $\{a(1,1), b(1), c(1,2)\}$.

We state the correctness of procedure $\mathcal{R}$ with the following theorem and proof.

Theorem 3.1 (Correctness)**.** *Let* Π *be any non-ground, tight program. Then, the grounding procedure* $\mathcal{R}$ *on* Π *is correct, i.e., the answer sets of* $\mathcal{R}(\Pi)$ *restricted to* $\mathrm{at}(\mathcal{G}(\Pi))$ *match the answer sets of* $\mathcal{G}(\Pi)$*. Precisely, for every answer set* M' *of* $\mathcal{R}(\Pi)$ *there is an answer set* $M' \cap \mathrm{at}(\mathcal{G}(\Pi))$ *of* $\mathcal{G}(\Pi)$.

Guess Answer Set Candidates

for every $h(X) \in \text{heads}(\Pi)$, $D \in \text{dom}(X)$:

$$h(D) \vee \bar{h}(D) \leftarrow \tag{$\mathcal{R}1$}$$

Ensure Satisfiability

for every $r \in \Pi$, $x \in \text{var}(r)$:

$$\bigvee_{d \in \text{dom}(x)} \text{sat}_x(d) \leftarrow \tag{$\mathcal{R}2$}$$

for every $r \in \Pi, p(X) \in B_r^+$, $D \in \text{dom}(X)$, $X = \langle x_1, \ldots, x_\ell \rangle$:

$$\text{sat}_r \leftarrow \text{sat}_{x_1}(D_{\langle x_1 \rangle}), \ldots, \text{sat}_{x_\ell}(D_{\langle x_\ell \rangle}), \neg p(D) \tag{$\mathcal{R}3$}$$

for every $r \in \Pi$, $p(X) \in H_r \cup B_r^-$, $D \in \text{dom}(X)$, $X = \langle x_1, \ldots, x_\ell \rangle$:

$$\text{sat}_r \leftarrow \text{sat}_{x_1}(D_{\langle x_1 \rangle}), \ldots, \text{sat}_{x_\ell}(D_{\langle x_\ell \rangle}), p(D) \tag{$\mathcal{R}4$}$$

where $\Pi = \{r_1, \ldots, r_n\}$:

$$\text{sat} \leftarrow \text{sat}_{r_1}, \ldots, \text{sat}_{r_n} \tag{$\mathcal{R}5$}$$

for every $r \in \Pi$, $x \in \text{var}(r)$, $d \in \text{dom}(x)$:

$$\text{sat}_x(d) \leftarrow \text{sat} \tag{$\mathcal{R}6$}$$

$$\leftarrow \neg\text{sat} \tag{$\mathcal{R}7$}$$

Prevent Unfoundedness

for every $r \in \Pi$, $h(X) \in H_r$, $D \in \text{dom}(X)$, $y \in \text{var}(r)$, $y \notin X$:

$$\bigvee_{d \in \text{dom}(y)} \text{uf}_y(\langle D, d \rangle) \leftarrow h(D) \tag{$\mathcal{R}8$}$$

for every $r \in \Pi$, $h(X) \in H_r$, $p(Y) \in B_r^+$, $D \in \text{dom}(\langle X, Y \rangle)$, $Y = \langle y_1, \ldots, y_\ell \rangle$:

$$\text{uf}_r(D_X) \leftarrow \text{uf}_{y_1}(D_{\langle X, y_1 \rangle}), \ldots, \text{uf}_{y_\ell}(D_{\langle X, y_\ell \rangle}), \neg p(D_Y) \tag{$\mathcal{R}9$}$$

for every $r \in \Pi$, $h(X) \in H_r$, $p(Y) \in B_r^- \cup (H_r \setminus \{h(X)\})$, $D \in \text{dom}(\langle X, Y \rangle)$, $Y = \langle y_1, \ldots, y_\ell \rangle$:

$$\text{uf}_r(D_X) \leftarrow \text{uf}_{y_1}(D_{\langle X, y_1 \rangle}), \ldots, \text{uf}_{y_\ell}(D_{\langle X, y_\ell \rangle}), p(D_Y) \tag{$\mathcal{R}10$}$$

for every $h(X) \in \text{heads}(\Pi)$, $D \in \text{dom}(X)$, $\{r_1, \ldots, r_m\} = \{r \in \Pi \mid h(Y) \in H_r\}$:

$$\leftarrow \text{uf}_{r_1}(D), \ldots, \text{uf}_{r_m}(D) \tag{$\mathcal{R}11$}$$

Figure 3.1: Body-decoupled grounding procedure $\mathcal{R}$ for a given non-ground, tight program Π, which creates a disjunctive *ground* program.

Proof. $\Leftarrow$: Let M be an answer set of $\mathcal{G}(\Pi)$ and assume towards a contradiction that there is no extension $M' \supseteq M$ of M with $M \cap \text{at}(\mathcal{G}(\Pi)) = M' \cap \text{at}(\mathcal{G}(\Pi))$ such that M' is an answer set of $\mathcal{R}(\Pi)$. First, we construct $N := \{\bar{h}(D) \mid h(X) \in \text{heads}(\Pi), D \in \text{dom}(X), h(D) \notin M\}$, which collects those head atoms that are not in M. Further, we gather satisfied rules by $S := \{\text{sat}, \text{sat}_r, \text{sat}_x(d) \mid r \in \Pi, x \in \text{var}(r), d \in \text{dom}(x)\}$. Then, we set appropriate domain values for the founded atoms $F := \{\text{uf}_{y_1}(\langle D, D'_{y_1} \rangle), \ldots, \text{uf}_{y_\ell}(\langle D, D'_{y_\ell} \rangle) \mid r \in \Pi, h(X) \in H_r, \{y \in \text{var}(r) \mid y \notin X\} = \{y_1, \ldots, y_\ell\}, D' \in \text{dom}(\langle \text{var}(r) \rangle), h(D) \in M, D_X =$

Guess Answer Set Candidates

$a(1,1) \vee \bar{a}(1,1).\ a(1,2) \vee \bar{a}(1,2).$ $(\mathcal{R}1)$

Ensure Satisfiability

$\mathrm{sat}_X(1).\ \mathrm{sat}_Y(1) \vee \mathrm{sat}_Y(2).\ \mathrm{sat}_Z(1) \vee \mathrm{sat}_Z(2).$ $(\mathcal{R}2)$

$\mathrm{sat}_r \leftarrow \mathrm{sat}_X(1), \neg b(1).$ $(\mathcal{R}3)$

$\mathrm{sat}_r \leftarrow \mathrm{sat}_Y(1), \mathrm{sat}_Z(1), \neg c(1,1).\ \mathrm{sat}_r \leftarrow \mathrm{sat}_Y(1), \mathrm{sat}_Z(1), \neg c(1,2).$

$\mathrm{sat}_r \leftarrow \mathrm{sat}_Y(2), \mathrm{sat}_Z(1), \neg c(2,1).\ \mathrm{sat}_r \leftarrow \mathrm{sat}_Y(2), \mathrm{sat}_Z(1), \neg c(2,2).$

$\mathrm{sat}_r \leftarrow \mathrm{sat}_X(1), \mathrm{sat}_Y(1), a(1,1).$ $(\mathcal{R}4)$

$\mathrm{sat}_r \leftarrow \mathrm{sat}_X(1), \mathrm{sat}_Y(2), a(1,2).$

$\mathrm{sat} \leftarrow \mathrm{sat}_r.$ $(\mathcal{R}5)$

$\mathrm{sat}_X(1) \leftarrow \mathrm{sat}.\ \mathrm{sat}_Y(1) \leftarrow \mathrm{sat}.\ \mathrm{sat}_Y(2) \leftarrow \mathrm{sat}.\ \mathrm{sat}_Z(1) \leftarrow \mathrm{sat}.\ \mathrm{sat}_Z(2) \leftarrow \mathrm{sat}.$ $(\mathcal{R}6)$

$\leftarrow \neg\mathrm{sat}.$ $(\mathcal{R}7)$

Prevent Unfoundedness

$\mathrm{uf}_Z(1,1,1) \vee \mathrm{uf}_Z(1,1,2) \leftarrow a(1,1).\ \mathrm{uf}_Z(1,2,1) \vee \mathrm{uf}_Z(1,2,2) \leftarrow a(1,2).$ $(\mathcal{R}8)$

$\mathrm{uf}_r(1,1) \leftarrow \neg b(1).\ \mathrm{uf}_r(1,2) \leftarrow \neg b(1).$ $(\mathcal{R}9)$

$\mathrm{uf}_r(1,1) \leftarrow \neg c(1,1), \mathrm{uf}_Z(1,1,1).\ \mathrm{uf}_r(1,1) \leftarrow \neg c(1,2), \mathrm{uf}_Z(1,1,2).$

$\mathrm{uf}_r(1,2) \leftarrow \neg c(2,1), \mathrm{uf}_Z(1,2,1).\ \mathrm{uf}_r(1,2) \leftarrow \neg c(2,2), \mathrm{uf}_Z(1,2,2).$

$\leftarrow a(1,1), \mathrm{uf}_r(1,1).\ \leftarrow a(1,2), \mathrm{uf}_r(1,2).$ $(\mathcal{R}11)$

Figure 3.2: $\mathcal{R}(\Pi)$ with Π from Example 3.2, guided by $\mathcal{R}$ of Figure 3.1.

$D'_X, p(Z) \in B_r^+ \cup B_r^-, (p(Z) \in B_r^+)$ iff $(p(D'_Z) \in M)\}$. Finally, we set the domain values for unfounded atoms $U := \{\mathrm{uf}_r(D), \mathrm{uf}_y(\langle D, d(y)\rangle) \mid r \in \Pi, h(X) \in H_r, h(D) \in M, y \in \mathrm{var}(r), y \notin X,$ there is no $\mathrm{uf}_y(\langle D, d'\rangle) \in F\}$, where $d(y)$ yields any arbitrary, fixed domain value in $\mathrm{dom}(y)$. Then, we let $M' := M \cup N \cup F \cup U$. Since by assumption M' is not an answer set, either (i) one of the rules is not satisfied, or (ii) M' is not minimal.

We proceed by case distinction. Case (i): Some rule $r \in \mathcal{R}(\Pi)$ is not satisfied by M'. It is easy to see that all heads of Rules $(\mathcal{R}1)$–$(\mathcal{R}8)$ are satisfied by construction of M'. Rules $(\mathcal{R}8)$ are satisfied by construction of F and U. The construction of U also ensures that Rules $(\mathcal{R}9)$ and $(\mathcal{R}10)$ are satisfied. Observe that Rules $(\mathcal{R}11)$ are satisfied, since M is an answer set of $\mathcal{G}(\Pi)$ and therefore every atom of M is founded, as constructed by F. This concludes Case (i), since every rule of $\mathcal{R}(\Pi)$ is satisfied.

Case (ii): Model M' of $\mathcal{R}(\Pi)$ is not minimal, i.e., there exists a model $M'' \subsetneq M'$ with M'' being a model of $\mathcal{R}(\Pi)^{M'}$. Note that the difference between M' and M'' cannot be due to Rule $(\mathcal{R}1)$, since then M' and M'' would be incomparable. If $\mathrm{sat} \notin M''$, then by Rules $(\mathcal{R}5)$, there is a rule $r_i \in \Pi$ with $\mathrm{sat}_{r_i} \notin M''$, which by Rules $(\mathcal{R}3)$ and $(\mathcal{R}4)$ implies that $\mathcal{G}(r_i)$ is not satisfied by M''. This, however, contradicts the assumption that M is

an answer set of $\mathcal{G}(\Pi)$. Consequently, by Rules ($\mathcal{R}6$), the difference between M' and M'' cannot be due to any sat_x predicate. Further, M' and M'' cannot differ in any predicate of the form uf_y either, since by Rules ($\mathcal{R}8$), for every $h(D) \in M'$, there is precisely one $\text{uf}_y(\langle D, d\rangle) \in M'$. Similarly, Rules ($\mathcal{R}9$)–($\mathcal{R}11$) yield deterministic consequences, depending only on the choice of Rules ($\mathcal{R}1$) and ($\mathcal{R}8$). As a result, we conclude that $M' = M''$ cannot differ.

$\Rightarrow$: Let M' be an answer set of $\mathcal{R}(\Pi)$ and assume towards a contradiction that $M := M' \cap \text{at}(\mathcal{G}(\Pi))$ is not an answer set of $\mathcal{G}(\Pi)$. Then, either (i) M is not a model of $\mathcal{G}(\Pi)$, or (ii) there is an $h(D) \in M$ that is unfounded by M. We proceed by case distinction.

Case (i): M is not a model of $\mathcal{G}(\Pi)$. Assume towards a contradiction that there is a rule $r_i \in \Pi$ with M not satisfying $\mathcal{G}(\{r_i\})$. Then, there is a subset $M'' \subsetneq M'$ with $\text{sat}_{r_i} \notin M''$, $\text{sat} \notin M''$ and sat_x predicate set accordingly for $x \in \text{var}(r_i)$, that is a model of $\mathcal{G}(\Pi)$, which contradicts that M' is an answer set ($\subseteq$-minimal model) of $\mathcal{R}(\Pi)$.

Case (ii): There exists an $h(D) \in M$ that is unfounded by M. Then, no matter how the predicates uf_x are assigned in M', since $h(D) \in M$ is not founded, there is no rule $r_i \in \Pi$ that can be instantiated such that the body is satisfied and thereby justifying $h(D)$. Consequently, for every such rule r_i, by Rules ($\mathcal{R}9$) and ($\mathcal{R}10$), there is at least one atom over predicates uf_{r_i} in M'. Finally, Rules ($\mathcal{R}11$) are not satisfied (for $h(D)$), which contradicts that M' is an answer set of $\mathcal{R}(\Pi)$. □

The procedure $\mathcal{R}$ works in polynomial time, since our technique does not suffer from large rules (or large rule bodies).

Theorem 3.2 (Polynomial Runtime and Grounding Size)**.** *Let* Π *be any non-ground, tight program, where every predicate has arity at most* a*. Then, the grounding procedure* $\mathcal{R}$ *on* Π *is polynomial, i.e., runs in time* $\mathcal{O}(\|\Pi\| \cdot |\text{dom}(\Pi)|^{2 \cdot a})$*.*

Proof. The reduction $\mathcal{R}$ constructs $\mathcal{O}(\|\Pi\| \cdot |\text{dom}(\Pi)|^a)$ many Rules ($\mathcal{R}1$) of constant size, $\mathcal{O}(\|\Pi\| \cdot a)$ many Rules ($\mathcal{R}2$) of size $|\text{dom}(\Pi)|$, as well as $\mathcal{O}(\|\Pi\| \cdot |\text{dom}(\Pi)|^a)$ many Rules ($\mathcal{R}3$) and ($\mathcal{R}4$) of size $\mathcal{O}(a)$. Then, there is one Rule ($\mathcal{R}5$) of size $\mathcal{O}(|\Pi|)$, $\mathcal{O}(\|\Pi\| \cdot |\text{dom}(\Pi)|)$ many Rules ($\mathcal{R}6$) and one Rule ($\mathcal{R}7$). Finally, there are $\mathcal{O}(\|\Pi\| \cdot |\text{dom}(\Pi)|^a \cdot a)$ many Rules ($\mathcal{R}8$) of size $\mathcal{O}(|\text{dom}(\Pi)| \cdot a)$, and we require $\mathcal{O}(\|\Pi\| \cdot |\text{dom}(\Pi)|^{2 \cdot a})$ effort for Rules ($\mathcal{R}9$) and ($\mathcal{R}10$), as well as $\mathcal{O}(\|\Pi\| \cdot |\text{dom}(\Pi)|)$ effort for Rules ($\mathcal{R}11$). □

We do not expect a significant runtime improvement in the worst case. Further, the expressiveness increase from normal to disjunctive programs is inevitable (already for fixed arity).

Proposition 3.1 (Disjunctive Programs Inevitable)**.** *Let* Π *be any non-ground, tight program, where every predicate has arity at most* a*. Then, unless* $\mathsf{NP} = \Sigma_2^P$*, there cannot be a polynomial grounding procedure* $\mathcal{R}'$*, where* $\mathcal{R}'(\Pi)$ *is normal.*

Proof. While consistency for ground, normal programs is in NP, Σ_2^P-hardness for disjunctive (head-cycle-free) non-ground programs, cf. [EFFW07, Lem. 6], can be lifted to non-ground, tight programs, by observing tightness after converting disjunctive rules into normal ones via shifting. □

3.1.1 Optimizations

The reduction $\mathcal{R}$, as described above, can also be further optimized. In the following we introduce optimizations that decrease the resulting ground rules and allow for more practicability.

Partial Reducability While the theory of decoupling rule bodies might seem tempting in theory, the performance of grounders that were highly optimized over years will probably not be achievable, especially for simpler problem encodings. Conscious of what optimized grounders are capable of, our approach can be optimized such that only tough rules are reduced by our reduction. Notably, with our approach the program can be split and the reduction $\mathcal{R}$ just partially applied, as long as the foundedness can still be provided. We back this statement with the following Corollary.

Corollary 3.1 (Partial Reducibility)**.** *Given a non-ground, tight program* Π *and a partition of* Π *into programs* Π_1, Π_2 *with* $\mathrm{hpreds}(\Pi_1) \cap \mathrm{hpreds}(\Pi_2) = \emptyset$*. Then, the answer sets of* $\mathcal{R}(\Pi_1) \cup \mathcal{G}(\Pi_2)$ *restricted to* $\mathrm{at}(\mathcal{G}(\Pi))$ *match those of* $\mathcal{G}(\Pi)$*.*

Proof (Idea). The result is a direct consequence of the proof construction of Theorem 3.1, which can be extended. Satisfiability checking of $\mathcal{R}(\Pi_1) \cup \mathcal{G}(\Pi_2)$ works similarly, since satisfiability of rules in $\mathcal{G}(\Pi_1)$ are ensured by Rules ($\mathcal{R}2$)–($\mathcal{R}7$), and satisfiability of rules $\mathcal{G}(\Pi_2)$ is treated directly.

Since $\mathrm{hpreds}(\Pi_1) \cap \mathrm{hpreds}(\Pi_2) = \emptyset$, predicates in $\mathrm{hpreds}(\Pi_1)$, $\mathrm{hpreds}(\Pi_2)$ can only appear in *rule bodies* of Π_2, Π_1, respectively. Consequently, for a given model of $\mathcal{G}(\Pi)$, the foundedness of atoms over predicates $\mathrm{hpreds}(\Pi_1)$ is decided by $\mathcal{R}(\Pi_1)$, namely Rules ($\mathcal{R}8$)–($\mathcal{R}11$), whereas foundedness of atoms over $\mathrm{hpreds}(\Pi_2)$ is checked by $\mathcal{G}(\Pi_2)$. □

Thereby, partial reducability not only allows for a more broad field of application, but also for interaction of sophisticated grounders and the reduction $\mathcal{R}$. As this implies, while the reduction can handle harder rules with large rule bodies, traditional grounders can still be used for simpler rule sets.

Example 3.3. *Consider the non-ground, tight program* $\Pi_2 := \Pi \cup \{r_2\}$ *with* $r_2 = d(X) \leftarrow b(X)$*, and program* Π *and facts* $\mathcal{F} = \{b(1).\ c(1,2).\}$ *from Example 3.2. One can split the program* Π_2 *into program parts* $\Pi_{2.1} := \{a(X,Y) \leftarrow b(X), c(Y,Z)\}$ *and* $\Pi_{2.2} := \{d(X) \leftarrow b(X)\}$ *to achieve partial reducability, since* $\mathrm{hpreds}(\Pi_{2.1}) \cap \mathrm{hpreds}(\Pi_{2.2}) = \emptyset$*. Then, when grounding subprogram* $\Pi_{2.1}$ *using the reduction* $\mathcal{R}$ *and* $\Pi_{2.2}$ *using a traditional grounder* $\mathcal{G}$*, the results* $\mathcal{R}(\Pi_{2.1})$ *and* $\mathcal{G}(\Pi_{2.2})$ *can be joined resulting in Figure 3.2 with an additional*

Improved Foundedness, replacing ($\mathcal{R}9$)–($\mathcal{R}11$) of $\mathcal{R}$

for every $r{\in}\Pi$, $h(X){\in}H_r$, $p(Y){\in}B_r^+$, $D{\in}\mathrm{dom}(\langle X,Y\rangle)$, $Y{=}\langle y_1,\ldots,y_\ell\rangle$:

$$\mathrm{uf}_{\mathrm{rch}(Y,X)}(D_{\langle \mathrm{rch}(Y,X)\rangle}) \leftarrow \mathrm{uf}_{y_1}(D_{\langle X,y_1\rangle}),\ldots,\mathrm{uf}_{y_\ell}(D_{\langle X,y_\ell\rangle}),\neg p(D_Y) \qquad (\mathcal{R}'9)$$

for every $r{\in}\Pi$, $h(X){\in}H_r$, $p(Y){\in}B_r^-\cup(H_r\setminus\{h(X)\})$, $D{\in}\mathrm{dom}(\langle X,Y\rangle)$, $Y{=}\langle y_1,\ldots,y_\ell\rangle$:

$$\mathrm{uf}_{\mathrm{rch}(Y,X)}(D_{\langle \mathrm{rch}(Y,X)\rangle}) \leftarrow \mathrm{uf}_{y_1}(D_{\langle X,y_1\rangle}),\ldots,\mathrm{uf}_{y_\ell}(D_{\langle X,y_\ell\rangle}),p(D_Y) \qquad (\mathcal{R}'10)$$

for every $h(X){\in}\mathrm{heads}(\Pi)$, $D{\in}\mathrm{dom}(X)$, $\{r_1,\ldots,r_m\}{=}\{r{\in}\Pi|h(Y){\in}H_r\}$:

$$\leftarrow h(D),[\bigvee_{p(Y)\in B_{r_1}} \mathrm{uf}_{\mathrm{rch}(Y,X)}(D_{\langle \mathrm{rch}(Y,X)\rangle})],\ldots,[\bigvee_{p(Y)\in B_{r_m}} \mathrm{uf}_{\mathrm{rch}(Y,X)}(D_{\langle \mathrm{rch}(Y,X)\rangle})] \qquad (\mathcal{R}'11)$$

Figure 3.3: More involved formalization of checking for unfounded atoms based on Observation 3.2, yielding alternative $\mathcal{R}'$ (cf. $\mathcal{R}$ of Figure 3.1).

rule for $\mathcal{G}(\Pi_{2.2}){=}d(1)\leftarrow b(1)$. *Now, when involving facts* $\mathcal{F}$*, one will end up with answer set* $\{a(1,1),b(1),c(1,2),d(1)\}$*, as expected.*

Utilizing Variable Independencies When analyzing the established grounding procedure $\mathcal{R}$ and examples (cf. Example 3.2), we can identify redundancies in the rule set for checking for unfoundedness of non-ground rules. Especially, in cases where variables of the head-predicate of the non-ground rule do not occur in some body predicate, multiple rules are produced to cover the whole possible ground atom set.

Based on that consideration, we provide an improvement of the grounding procedure $\mathcal{R}$ utilizing reachable paths of the *variable graph*, which might significantly reduce the number of atoms for the foundedness check. To do so, we define the variable graph $\mathcal{V}_\Pi$ of a non-ground program Π in the following.

Definition 3.1 (Variable Graph)**.** *The variable graph* $\mathcal{V}_\Pi$ *of* Π *is the graph defined on the set* $\mathrm{var}(r)$ *of variables of rules* $r\in\Pi$*, where two variables* $x,y\in\mathrm{var}(r)$ *are joined by an edge* (x,y) *whenever there is a predicate* $p(X)$ *in* r *with* $x,y\in X$*. Let* X,Y *be variable vectors. We refer by* $\mathrm{rch}(Y,X)$ *to those vertices in* X *that are* reachable *by some* $y\in Y$ *in* $\mathcal{V}_\Pi$*.*

To this end, we rely on the following observation.

Observation 3.2 (Variable-Justification Independency)**.** *Given a non-ground program* Π*, any rule* $r\in\Pi$ *with* $h(X)\in H_r$ *and* $p(Y)\in B_r^+$*. Further, let* I *be a set of atoms over* $\mathcal{G}(\Pi)$*. Then, if* r *does not justify* $h(D_X)\in I$ *for* $D\in\mathrm{dom}(\langle X,Y\rangle)$ *due to* $p(D_Y)\notin I$*, we have that* r *fails to justify* any $h(D')\in I$ *with* $D'\in\mathrm{dom}(X)$ *and* $D'_{\langle \mathrm{rch}(Y,X)\rangle}=D_{\langle \mathrm{rch}(Y,X)\rangle}$ *as well.*

Based on the observation above, we provide an alternative $\mathcal{R}'$ of reduction $\mathcal{R}$ (see Figure 3.3), which is optimized to consider independent variables according to Observa-

Improved Unfoundedness

$\mathrm{uf}_{\{X\}}(1) \leftarrow \neg b(1).\ \mathrm{uf}_{\{X\}}(2) \leftarrow \neg b(2).$ $(\mathcal{R}'9)$

$\mathrm{uf}_{\{Y\}}(1) \leftarrow \neg c(1,1), \mathrm{uf}_Z(1,1).\ \mathrm{uf}_{\{Y\}}(1) \leftarrow \neg c(1,2), \mathrm{uf}_Z(1,2).$

$\mathrm{uf}_{\{Y\}}(2) \leftarrow \neg c(2,1), \mathrm{uf}_Z(2,1).\ \mathrm{uf}_{\{Y\}}(2) \leftarrow \neg c(2,2), \mathrm{uf}_Z(2,2).$

$\leftarrow a(1,1), [\mathrm{uf}_{\{X\}}(1) \vee \mathrm{uf}_{\{Y\}}(1)].$ $(\mathcal{R}'11)$

$\leftarrow a(1,2), [\mathrm{uf}_{\{X\}}(1) \vee \mathrm{uf}_{\{Y\}}(2)].$

Figure 3.4: Optimized foundedness check using the reduction $\mathcal{R}'$ for Π from Example 3.2.

tion 3.2. Instead of AtUf, we use atoms $\mathrm{AtUfI} := \{\mathrm{uf}_{\mathrm{rch}(Y,X)}(D_X), \mathrm{uf}_y(D_{\langle X,y\rangle}) | r \in \Pi, D \in \mathrm{dom}(\langle \mathrm{var}(r)\rangle), h(X) \in H_r, p(Y) \in B_r^+ \cup B_r^-, y \in Y, y \notin X\}$. The updated reduction $\mathcal{R}'$ is given in Figure 3.3, where Rules $(\mathcal{R}'9)$ and $(\mathcal{R}'10)$ replace Rules $(\mathcal{R}9)$ and $(\mathcal{R}10)$ with the only difference that a different head predicate is used with a potentially smaller domain vector. Further, Rules $(\mathcal{R}11)$ are replaced by Rules $(\mathcal{R}'11)$, which contain disjunctions in their bodies, as in the well-known weight rules [GKS11]. Alternatively, one can build plain rules by creating the cross product among the sets of disjuncts of Rules $(\mathcal{R}'11)$.

Example 3.4. *Consider the non-ground, tight program Π of Example 3.2. Based on the Observation 3.2, the rules for preventing unfoundedness can be improved and reduced by incorporating variable independencies, resulting in less rules for ensuring foundedness, as shown in Figure 3.4.*

Comparing to Figure 3.2, one can identify that Rules $(\mathcal{R}'9)$-$(\mathcal{R}'11)$ do in fact reduce the number of necessary instantiations and rules. While the head-predicate of the only rule r in the program Π does contain the two variables X and Y, they are only partly incorporated in the (decoupled) body-predicates b and c of r. Therefore, the rule set $(\mathcal{R}'9)$ can be reduced to the domain vector only containing variable X when checking for predicate b and, similar, to only Y when checking for c. This way the number of instantiations can be reduced to the combinations of variables actually affecting the body-predicate. To incorperate these changes and the different occurrences of variables in the head-predicate $\mathrm{uf}_{\mathrm{rch}}$ of Rules $(\mathcal{R}'9)$ and $(\mathcal{R}'10)$, we switch in Rules $(\mathcal{R}'11)$ to weighted disjunctions of these predicates in the body.

3.2 Body-Decoupled Grounding for Normal ASP

The idea of our reduction $\mathcal{R}'$ can be also lifted to non-ground, normal programs. To this end, one can rely on orderings (level mappings), as defined in Section 2.2.1. However, to simplify the presentation, we only show a simplified variant that uses a quadratic number of auxiliary atoms for comparison, instead of encoding orderings. We therefore use for every two distinct ground atoms $p(D)$, $p'(D)$ of Π, an additional auxiliary

Additional Rules for Foundedness of Normal Programs

for every $p(X), p'(X') \in \text{heads}(\Pi)$, $D \in \text{dom}(X)$, $D' \in \text{dom}(X')$, $p(D) \neq p'(D')$:

$$[p(D) \prec p'(D')] \vee [p'(D') \prec p(D)] \leftarrow \tag{$\mathcal{R}''12$}$$

for every $p_1(X_1), p_2(X_2), p_3(X_3) \in \text{heads}(\Pi)$, $D_1 \in \text{dom}(X_1)$, $D_2 \in \text{dom}(X_2)$, $D_3 \in \text{dom}(X_3)$, $p_1(D_1) \neq p_2(D_2)$, $p_1(D_1) \neq p_3(D_3)$, $p_2(D_2) \neq p_3(D_3)$:

$$\leftarrow [p_1(D_1) \prec p_2(D_2)], [p_2(D_2) \prec p_3(D_3)], [p_3(D_3) \prec p_1(D_1)] \tag{$\mathcal{R}''13$}$$

for every $r \in \Pi$, $h(X) \in H_r$, $p(Y) \in B_r^+$, $D \in \text{dom}(\langle X, Y \rangle)$, $Y = \langle y_1, \ldots, y_\ell \rangle$, $p(D_Y) \notin \mathcal{F}$:

$$\text{uf}_r(D_X) \leftarrow \text{uf}_{y_1}(D_{\langle X, y_1 \rangle}), \ldots, \text{uf}_{y_\ell}(D_{\langle X, y_\ell \rangle}), \neg[p(D_Y) \prec h(D_X)] \tag{$\mathcal{R}''14$}$$

Figure 3.5: For non-ground, normal programs, we require *additional rules* for ensuring foundedness, yielding $\mathcal{R}''$.

predicate $[p(D) \prec p'(D')]$ responsible for storing precedence in the order of derivation. Then, given $\mathcal{R}$ of Figure 3.1 (and the optimized reduction $\mathcal{R}'$ of Figure 3.3), for normal programs we just need to add those rules of Figure 3.5, yielding $\mathcal{R}''$ shown in Figure 3.5. Intuitively, Rules ($\mathcal{R}''12$) determine precedence among different atoms and Rules ($\mathcal{R}''13$) take care of transitivity of "$\prec$". In addition to Rules ($\mathcal{R}'9$) and ($\mathcal{R}'10$), Rules ($\mathcal{R}''14$) add a further case of unfoundedness, if precedence among atoms is not suitable for justifying foundedness.

Example 3.5. *Consider the non-ground, normal program* $\Pi_3 := \Pi \cup \{r_3\}$ *with* $r_3 = c(X, Y) \leftarrow a(X, Y)$, *and program* Π *and facts* $\mathcal{F} = \{b(1).\ c(1, 2).\}$ *from Example 3.2. While the grounding procedure* $\mathcal{R}$ *can be used identical for normal programs, the additional rules, shown in Figure 3.6 (Additional Rules for Foundedness of Normal Programs), ensure orderings as described above. For storing the order of derivation the auxiliary predicate* p *is used. Notice that the number of rules were kept minimal by using only the relevant variables per rule, i.e.* X, Y, Z *for* r *and* X, Y *for* r_2, *and taking the domain of each variable into consideration. The answer sets of* $P_3 = \mathcal{R}''(\Pi_3)$ *restricted to symbols* a, b, c *of* Π_3 *yield* $\{a(1, 1), b(1), c(1, 1), c(1, 2)\}$, *as expected.*

Analogously to Theorem 3.1, one can show correctness of $\mathcal{R}''$ for non-ground, normal programs, including when applied to program parts as in Corollary 3.1. To this end, we capture sufficient conditions of predicate dependencies as follows.

Definition 3.2 (Strongly-connected Component)**.** *Based on the dependency graph* $\mathcal{D}_\Pi$ *of a non-ground program* Π *(see Definition 2.6), a set* $C \subseteq \text{hpreds}(\Pi)$ *of predicates is a* strongly-connected component (SCC) *if* C *is a* $\subseteq$*-largest set such that for every two distinct predicates* p, q *in* C *there is a directed path from* p *to* q *in* $\mathcal{D}_\Pi$.

Based on that definition, SCCs of the dependency graph can be utilized for partially applying our reduction $\mathcal{R}''$.

Guess Answer Set Candidates

$a(1,1) \vee \bar{a}(1,1).\ a(1,2) \vee \bar{a}(1,2).$ $(\mathcal{R}1)$

Ensure Satisfiability

$\mathrm{sat}_X(1).\ \mathrm{sat}_Y(1) \vee \mathrm{sat}_Y(2).\ \mathrm{sat}_Z(1) \vee \mathrm{sat}_Z(2).$ $(\mathcal{R}2)$

$\mathrm{sat}_r \leftarrow \mathrm{sat}_X(1), \neg b(1).$ $(\mathcal{R}3)$

$\mathrm{sat}_r \leftarrow \mathrm{sat}_Y(1), \mathrm{sat}_Z(1), \neg c(1,1).\ \mathrm{sat}_r \leftarrow \mathrm{sat}_Y(1), \mathrm{sat}_Z(1), \neg c(1,2).$

$\mathrm{sat}_r \leftarrow \mathrm{sat}_Y(2), \mathrm{sat}_Z(1), \neg c(2,1).\ \mathrm{sat}_r \leftarrow \mathrm{sat}_Y(2), \mathrm{sat}_Z(1), \neg c(2,2).$

$\mathrm{sat}_r \leftarrow \mathrm{sat}_X(1), \mathrm{sat}_Y(1), a(1,1).$ $(\mathcal{R}4)$

$\mathrm{sat}_r \leftarrow \mathrm{sat}_X(1), \mathrm{sat}_Y(2), a(1,2).$

$\mathrm{sat} \leftarrow \mathrm{sat}_r.$ $(\mathcal{R}5)$

$\mathrm{sat}_X(1) \leftarrow \mathrm{sat}.\ \mathrm{sat}_Y(1) \leftarrow \mathrm{sat}.\ \mathrm{sat}_Y(2) \leftarrow \mathrm{sat}.\ \mathrm{sat}_Z(1) \leftarrow \mathrm{sat}.\ \mathrm{sat}_Z(2) \leftarrow \mathrm{sat}.$ $(\mathcal{R}6)$

$\leftarrow \neg\mathrm{sat}.$ $(\mathcal{R}7)$

Improved Unfoundedness

$\mathrm{uf}_{\{X\}}(1) \leftarrow \neg b(1).\ \mathrm{uf}_{\{X\}}(2) \leftarrow \neg b(2).$ $(\mathcal{R}'9)$

$\mathrm{uf}_{\{Y\}}(1) \leftarrow \neg c(1,1), \mathrm{uf}_Z(1,1).\ \mathrm{uf}_{\{Y\}}(1) \leftarrow \neg c(1,2), \mathrm{uf}_Z(1,2).$

$\mathrm{uf}_{\{Y\}}(2) \leftarrow \neg c(2,1), \mathrm{uf}_Z(2,1).\ \mathrm{uf}_{\{Y\}}(2) \leftarrow \neg c(2,2), \mathrm{uf}_Z(2,2).$

$\leftarrow a(1,1), [\mathrm{uf}_{\{X\}}(1) \vee \mathrm{uf}_{\{Y\}}(1)].$ $(\mathcal{R}'11)$

$\leftarrow a(1,2), [\mathrm{uf}_{\{X\}}(1) \vee \mathrm{uf}_{\{Y\}}(2)].$

Additional Rules for Foundedness of Normal Programs

$\mathrm{p}_{ac}(1,1) \vee \mathrm{p}_{ca}(1,1,1).\ \mathrm{p}_{ac}(1,1) \vee \mathrm{p}_{ca}(1,1,2).$ $(\mathcal{R}''12)$

$\mathrm{p}_{ac}(1,2) \vee \mathrm{p}_{ca}(1,2,1).\ \mathrm{p}_{ac}(1,2) \vee \mathrm{p}_{ca}(1,2,2).$

$\mathrm{uf}_r(1,1) \leftarrow \neg\mathrm{p}_{ca}(1,1,1), \mathrm{uf}_Z(1,1,1).$ $(\mathcal{R}''14)$

$\mathrm{uf}_r(1,1) \leftarrow \neg\mathrm{p}_{ca}(1,1,2), \mathrm{uf}_Z(1,1,2).$

$\mathrm{uf}_r(1,2) \leftarrow \neg\mathrm{p}_{ca}(1,2,1), \mathrm{uf}_Z(1,2,1).$

$\mathrm{uf}_r(1,2) \leftarrow \neg\mathrm{p}_{ca}(1,2,2), \mathrm{uf}_Z(1,2,2).$

$\mathrm{uf}_{r_2}(1,1) \leftarrow \neg\mathrm{p}_{ac}(1,1).\ \mathrm{uf}_{r_2}(1,2) \leftarrow \neg\mathrm{p}_{ac}(1,2).$

Figure 3.6: Ground program $P_3{=}\mathcal{R}''(\Pi_3)$ with Π from Example 3.5, guided by $\mathcal{R}$ of Figure 3.1 with optimizations of Figure 3.3 and additional rules of Figure 3.5.

Theorem 3.3 (Partial Reducibility/Normal Programs)**.** *Given a non-ground, normal program* Π *and a partition of* Π *into programs* Π_1, Π_2 *with* $\mathrm{hpreds}(\Pi_1) \cap \mathrm{hpreds}(\Pi_2) = \emptyset$*, where for every SCC* C_1 *of* $\mathcal{D}_{\Pi_1}$ *and SCC* C_2 *of* $\mathcal{D}_{\Pi_2}$*, we have* $C_1 \cap C_2 = \emptyset$*. Then, the answer sets of* $\mathcal{R}''(\Pi_1) \cup \mathcal{G}(\Pi_2)$ *restricted to* $\mathrm{at}(\mathcal{G}(\Pi))$ *match those of* $\mathcal{G}(\Pi)$*.*

Proof. $\Leftarrow$: Let M be an answer set of $\mathcal{G}(\Pi)$ and assume towards a contradiction

that there is no extension $M' \supseteq M$ of M with $M \cap \mathrm{at}(\mathcal{G}(\Pi)) = M' \cap \mathrm{at}(\mathcal{G}(\Pi))$ such that M' is an answer set of $\mathcal{R}''(\Pi_1) \cup \mathcal{G}(\Pi_2)$. Since M is an answer set of $\mathcal{G}(\Pi)$, there is an order $\varphi : M \to \{0, \ldots, |M| - 1\}$ that is used for showing foundedness of M. Similar to the proof of Theorem 3.1, we collect those head atoms not in M by $N := \{\bar{h}(D) \,|\, h(X) \in \mathrm{heads}(\Pi_1), D \in \mathrm{dom}(X), h(D) \notin M\}$. Then, we gather satisfied rules by $S := \{\mathrm{sat}, \mathrm{sat}_r, \mathrm{sat}_x(d) \mid r \in \Pi_1, x \in \mathrm{var}(r), d \in \mathrm{dom}(x)\}$. We set appropriate domain values for the founded atoms $F := \{\mathrm{uf}_{y_1}(\langle D, D'_{y_1}\rangle), \ldots, \mathrm{uf}_{y_\ell}(\langle D, D'_{y_\ell}\rangle) \mid r \in \Pi_1, h(X) \in H_r, \{y \in \mathrm{var}(r) \,|\, y \notin X\} = \{y_1, \ldots, y_\ell\}, D' \in \mathrm{dom}(\langle \mathrm{var}(r)\rangle), h(D) \in M, D_X = D'_X, p(Z) \in B_r^+ \cup B_r^-, (p(Z) \in B_r^+)$ iff $(p(D'_Z) \in M), [p(Z) \in B_r^-]$ or $[\varphi(p(D'_Z)) < \varphi(h(D))]\}$. Then, we set domain values for unfounded atoms $U := \{\mathrm{uf}_r(D), \mathrm{uf}_y(\langle D, d(y)\rangle) \mid r \in \Pi_1, h(X) \in H_r, h(D) \in M, y \in \mathrm{var}(r), y \notin X,$ there is no $\mathrm{uf}_y(\langle D, d'\rangle) \in F\}$, where $d(y)$ yields any arbitrary, fixed domain value in $\mathrm{dom}(y)$.

We encode ordering φ by setting $O := \{[a \prec b] \mid \{a, b\} \subseteq M, \varphi(a) < \varphi(b)\}$ and adding those over atoms not only in M by $P := \{[p(D) \prec q(D')] \mid \{p(X), q(Y)\} \subseteq \mathrm{heads}(\Pi_1), p(D) \in \mathrm{dom}(X), q(D') \in \mathrm{dom}(Y), p(D) \notin M$ or $q(D') \notin M, q(D') \succ p(D)\}$, where $\succ$ is any arbitrary total ordering over atoms $\mathrm{at}(\mathcal{G}(\Pi_1))$. Finally, we let $M' := M \cup N \cup F \cup U \cup O \cup P$. Since by assumption M' is not an answer set, either (i) one of the rules is not satisfied, or (ii) M' is not minimal.

We proceed by case distinction. Case (i): Some rule $r \in \mathcal{R}''(\Pi_1) \cup \mathcal{G}(\Pi_2)$ is not satisfied by M'. Obviously $\mathcal{G}(\Pi_2)$ is satisfied since M is an answer set of $\mathcal{G}(\Pi)$. Further, it is easy to see that all heads of Rules ($\mathcal{R}1$)–($\mathcal{R}8$) are satisfied by construction of M'. Rules ($\mathcal{R}8$) are satisfied by construction of F and U. The construction of U also ensures that Rules ($\mathcal{R}9$) and ($\mathcal{R}10$) are satisfied. Observe that Rules ($\mathcal{R}11$) are satisfied, since M is an answer set of $\mathcal{G}(\Pi_1)$ and therefore every atom of M is founded, as constructed by F. Further, by construction of O and P, Rules ($\mathcal{R}''12$), ($\mathcal{R}''13$), and ($\mathcal{R}''14$) are satisfied. This concludes Case (i), since every rule of $\mathcal{R}''(\Pi_1) \cup \mathcal{G}(\Pi_2)$ is satisfied.

Case (ii): Model M' of $\mathcal{R}''(\Pi_1) \cup \mathcal{G}(\Pi_2)$ is not minimal, i.e., there exists a model $M'' \subsetneq M'$ with M'' being a model of $(\mathcal{R}''(\Pi_1) \cup \mathcal{G}(\Pi_2))^{M'}$. Note that the difference between M' and M'' cannot be due to Rule ($\mathcal{R}1$), since then M' and M'' would be incomparable. If $\mathrm{sat} \notin M''$, then by Rules ($\mathcal{R}5$), there is a rule $r_i \in \Pi$ with $\mathrm{sat}_{r_i} \notin M''$, which by Rules ($\mathcal{R}3$) and ($\mathcal{R}4$) implies that $\mathcal{G}(r_i)$ is not satisfied by M''. This, however, contradicts the assumption that M is an answer set of $\mathcal{G}(\Pi)$. Consequently, by Rules ($\mathcal{R}6$), the difference between M' and M'' cannot be due to any sat_x predicate. Further, M' and M'' cannot differ in any predicate of the form uf_y either, since by Rules ($\mathcal{R}8$), for every $h(D) \in M'$, there is precisely one $\mathrm{uf}_y(\langle D, d\rangle) \in M'$. Similarly, Rules ($\mathcal{R}9$)–($\mathcal{R}11$) yield deterministic consequences, depending only on the choice of Rules ($\mathcal{R}1$) and ($\mathcal{R}8$). Rules ($\mathcal{R}''12$) cannot be responsible for the difference between M' and M'', since then M' and M'' would be incomparable (every answer set has to contain precisely one of these head atoms per grounding of Rules ($\mathcal{R}''13$)). Finally, Rules ($\mathcal{R}''14$) also yield deterministic consequences, given the choice of Rules ($\mathcal{R}8$) and ($\mathcal{R}''14$). As a result, we conclude that M' and M'' can only differ due to Π_2. Then, however, $M'' \cap \mathrm{at}(\mathcal{G}(\Pi))$ is

also a model of $\mathcal{G}(\Pi_2)^M$. Further, $M'' \cap \text{at}(\mathcal{G}(\Pi))$ is a model of $\mathcal{G}(\Pi)^M$ as well, since M'' is model of $(\mathcal{R}''(\Pi_1) \cup \mathcal{G}(\Pi_2))^{M'}$, which by $\text{heads}(\Pi_1) \cap \text{heads}(\Pi_2) = \emptyset$, could only be prevented by atoms over body predicates of Π_1. Then, however, M is not an answer set since $M'' \cap \text{at}(\mathcal{G}(\Pi)) \subsetneq M$.

$\Rightarrow$: Let M' be an answer set of $\mathcal{R}''(\Pi_1) \cup \mathcal{G}(\Pi_2)$ and assume towards a contradiction that $M := M' \cap \text{at}(\mathcal{G}(\Pi))$ is not an answer set of $\mathcal{G}(\Pi)$. Then, either (i) M is not a model of $\mathcal{G}(\Pi)$, or (ii) there is an $h(D) \in M$ that is unfounded by M. We proceed by case distinction.

Case (i): M is not a model of $\mathcal{G}(\Pi)$. Assume towards a contradiction that there is a rule $r_i \in \Pi$ with M not satisfying $\mathcal{G}(\{r_i\})$. Then, $r_i \in \Pi_1$ since M is by definition a model of $\mathcal{G}(\Pi_2)$. Then, there is a subset $M'' \subsetneq M'$ with $\text{sat}_{r_i} \notin M''$, $\text{sat} \notin M''$ and sat_x predicate set accordingly for $x \in \text{var}(r_i)$, that is a model of $\mathcal{G}(\Pi)$, which contradicts that M' is an answer set ($\subseteq$-minimal model) of $\mathcal{R}''(\Pi_1) \cup \mathcal{G}(\Pi_2)$.

Case (ii): There exists an $h(D) \in M$ that is unfounded by $\mathcal{R}''(\Pi_1) \cup \mathcal{G}(\Pi_2)$. If $h \in \text{heads}(\Pi_2)$, then by construction of M, $h(D) \in M'$ is also unfounded by every rule in Π_2 (as well as by every rule in Π_1 since $\text{heads}(\Pi_1) \cap \text{heads}(\Pi_2) = \emptyset$ and there is no SCC overlap that could cause foundedness), contradicting that M' is an answer set. Otherwise, irrelevant how the predicates uf_x or predicates of the form $\cdot \prec \cdot$ are assigned in M', since $h(D) \in M$ is not founded, there is no rule $r_i \in \Pi_1$ that can be grounded such that the resulting body is satisfied and the ground rule justifies $h(D)$. Consequently, for every such rule r_i, by Rules ($\mathcal{R}9$), ($\mathcal{R}10$), and ($\mathcal{R}''14$) there is at least one atom over predicates uf_{r_i} in M'. Finally, Rules ($\mathcal{R}11$) are not satisfied (for $h(D)$), which contradicts that M' is an answer set of $\mathcal{R}''(\Pi_1) \cup \mathcal{G}(\Pi_2)$. □

In comparison to the procedure $\mathcal{R}'$, which runs in time $\mathcal{O}(\|\Pi\| \cdot |\text{dom}(\Pi)|^{2 \cdot a})$, we see from the runtime complexity that the reduction $\mathcal{R}''$ still runs in polynomial time, but might suffer from the ordering encodings.

Theorem 3.4 (Polynomial Runtime and Grounding Size)**.** *Let* Π *be any non-ground, normal program, where every predicate has arity at most* a*. Then, the grounding procedure* $\mathcal{R}''$ *on* Π *is polynomial, i.e., runs in time* $\mathcal{O}((\|\Pi\| \cdot |\text{dom}(\Pi)|^a)^3)$.

Proof. The reduction $\mathcal{R}''$ is constructed similar to reduction $\mathcal{R}$, for which we have shown a runtime of $\mathcal{O}(\|\Pi\| \cdot |\text{dom}(\Pi)|^{2 \cdot a})$. The additional steps of the reduction construct $\mathcal{O}((\|\Pi\| \cdot |\text{dom}(\Pi)|^a)^2)$ many Rules ($\mathcal{R}''12$) of constant size, $\mathcal{O}((\|\Pi\| \cdot |\text{dom}(\Pi)|^a)^3)$ many Rules ($\mathcal{R}''13$) of constant size and $\mathcal{O}(\|\Pi\| \cdot |\text{dom}(\Pi)|^2 \cdot a)$ many Rules ($\mathcal{R}''14$) of size $\mathcal{O}(a)$. □

3.3 Body-Decoupled Grounding Beyond Normal ASP

In this section the idea of grounding via reduction is pursued beyond normal logic programs, i.e. to disjunctive (or full) logic programs. In comparison to the above

introduced reductions, the approach is considering ground Elp programs instead of Asp programs, which is motivated in the following.

Motivation & Differences While the motivation for body-decoupled grounding as described above still applies, Table 2.1 shows, the complexity of ground Asp programs does not allow for a similar reduction (as in Sections 3.1 and 3.2) from disjunctive programs since its complexity is limited to Σ_2^P, whereas (full) non-ground programs are known to be Σ_3^P (when assuming bounded predicate arity). We therefore require a problem of similar complexity for which solvers are available and to which we can reduce (efficiently) from non-ground programs.

As defined in Section 2.3, epistemic logic programs implement a way to go beyond the expressive power of Asp programs. Elp presents an extension to standard Asp and allows reasoning over multiple worlds, i.e. deriving certain consequences depending on whether objections are known or possible. With this increase in expressiveness and the enabling of encoding harder problems, the complexity of (ground) Elp ascends to Σ_3^P completeness [SE16]. This lies in consensus with the complexity of non-ground Asp programs, which seems to be fitting for our reduction approach. Further, since Elp presents an extension to standard logic programs an efficient reduction from Asp and to Elp seems obvious.

Interestingly, epistemic logic programs were subject to many recent works in the last years. Especially, much attention was paid to efficiently solving these programs. In particular, we list approaches using reductions [BMW20], multi-shot solving as an extension to clingo [CFG+20] and hybrid solving techniques that utilize treewidth to break down the effort of solving [BHW21]. These improvements make Elp an attractive problem to work with.

Based on these observations, we present our grounding approach for non-ground, disjunctive programs by reducing to ground Elp. While yielded an epistemic extension to logic programs, the encoding is still constructed in a way that the advantages of body-decoupling are still achieved.

Body-Decoupled Grounding for Disjunctive Programs Similar to the earlier reductions, we assume a given non-ground program Π and a set $\mathcal{F}$ of facts. As for the approach for normal programs, we use every instantiation of $p(X)$ and its negation $\dot{p}(X)$ over $\mathrm{dom}(\Pi)$ for each predicate $p(X)$ in $\mathrm{heads}(\Pi)$, resulting in atoms $\mathrm{AtPred} := \{p(D), \dot{p}(D) \mid p(X) \in \mathrm{heads}(\Pi), D \in \mathrm{dom}(X)\}$. Further, we define $\mathrm{AtPredC} := \{p^{\mathsf{c}}(X), \dot{p}^{\mathsf{c}}(X) \mid p(X) \in \mathrm{AtPred}\}$.

In the accordance with the semantics of Asp, we require to ensure (i) satisfiability of a potential model M of Π and (ii) non-existence of counter witness, i.e. a model $C \subset M$ (satisfying (i)) of Π^M. Notice that, by performing subset minimization, we do not require a foundedness check as in earlier versions. For (i) computing models of rules, we require atoms $\mathrm{AtSat}^{\mathsf{1}} := \{\mathrm{sat}^{\mathsf{1}}, \mathrm{sat}_r^{\mathsf{1}}, \mathrm{gr}_x^{\mathsf{1}}(d) \mid r \in \Pi, x \in \mathrm{var}(r), d \in \mathrm{dom}(x)\}$, where $\mathrm{sat}^{\mathsf{1}}$ (sat_r)

Ensure Satisfiability of Answer-Set Candidates

for every $r \in \Pi$, $x \in \mathrm{var}(r)$:

$$\bigvee_{d \in \mathrm{dom}(x)} \mathrm{gr}_x(d) \leftarrow \tag{S1}$$

for every $r \in \Pi$, $p(X) \in B_r^+$, $D \in \mathrm{dom}(X)$, $X = \langle x_1, \ldots, x_\ell \rangle$:

$$\mathrm{sat}_r \leftarrow \mathrm{gr}_{x_1}(D_{\langle x_1 \rangle}), \ldots, \mathrm{gr}_{x_\ell}(D_{\langle x_\ell \rangle}), \neg p(D) \tag{S2}$$

for every $r \in \Pi$, $p(X) \in H_r \cup B_r^-$, $D \in \mathrm{dom}(X)$, $X = \langle x_1, \ldots, x_\ell \rangle$:

$$\mathrm{sat}_r \leftarrow \mathrm{gr}_{x_1}(D_{\langle x_1 \rangle}), \ldots, \mathrm{gr}_{x_\ell}(D_{\langle x_\ell \rangle}), p(D) \tag{S3}$$

where $\Pi = \{r_1, \ldots, r_n\}$:

$$\mathrm{sat} \leftarrow \mathrm{sat}_{r_1}, \ldots, \mathrm{sat}_{r_n} \tag{S4}$$

for every $r \in \Pi$, $x \in \mathrm{var}(r)$, $d \in \mathrm{dom}(x)$:

$$\mathrm{gr}_x(d) \leftarrow \mathrm{sat} \tag{S5}$$

Ensure Satisfiability of $\subseteq$-Smaller Models

for every $r \in \Pi$, $x \in \mathrm{var}(r)$:

$$\bigvee_{d \in \mathrm{dom}(x)} \mathrm{gr}^{\mathsf{c}}_x(d) \leftarrow \tag{S6}$$

for every $r \in \Pi$, $p(X) \in B_r^+$, $D \in \mathrm{dom}(X)$, $X = \langle x_1, \ldots, x_\ell \rangle$:

$$\mathrm{sat}^{\mathsf{c}}_r \leftarrow \mathrm{gr}^{\mathsf{c}}_{x_1}(D_{\langle x_1 \rangle}), \ldots, \mathrm{gr}^{\mathsf{c}}_{x_\ell}(D_{\langle x_\ell \rangle}), \neg p^{\mathsf{c}}(D) \tag{S7}$$

for every $r \in \Pi$, $p(X) \in H_r$, $D \in \mathrm{dom}(X)$, $X = \langle x_1, \ldots, x_\ell \rangle$:

$$\mathrm{sat}^{\mathsf{c}}_r \leftarrow \mathrm{gr}^{\mathsf{c}}_{x_1}(D_{\langle x_1 \rangle}), \ldots, \mathrm{gr}^{\mathsf{c}}_{x_\ell}(D_{\langle x_\ell \rangle}), p^{\mathsf{c}}(D) \tag{S8}$$

for every $r \in \Pi$, $p(X) \in B_r^-$, $D \in \mathrm{dom}(X)$, $X = \langle x_1, \ldots, x_\ell \rangle$:

$$\mathrm{sat}^{\mathsf{c}}_r \leftarrow \mathrm{gr}^{\mathsf{c}}_{x_1}(D_{\langle x_1 \rangle}), \ldots, \mathrm{gr}^{\mathsf{c}}_{x_\ell}(D_{\langle x_\ell \rangle}), p(D) \tag{S9}$$

where $\Pi = \{r_1, \ldots, r_n\}$:

$$\mathrm{sat}^{\mathsf{c}} \leftarrow \mathrm{nempty}, \mathrm{sat}^{\mathsf{c}}_{r_1}, \ldots, \mathrm{sat}^{\mathsf{c}}_{r_n} \tag{S10}$$

for every $r \in \Pi$, $x \in \mathrm{var}(r)$, $d \in \mathrm{dom}(x)$:

$$\mathrm{gr}^{\mathsf{c}}_x(d) \leftarrow \mathrm{sat}^{\mathsf{c}} \tag{S11}$$

Figure 3.7: Reduction $\mathsf{S}(\Pi)$ for encoding satisfiablity of a non-ground program Π into a ground program requiring the use of disjunction through Formulas (S1) and (S6).

indicates satisfiability (of non-ground rule r) for a potential model $\mathtt{l} \in \{\epsilon, \mathsf{c}\}$, where we use ϵ for model M and c for a potentially smaller model N. An atom of the form $\mathrm{gr}^{\mathtt{l}}_x(d)$ indicates that for checking satisfiability, we assign variable x of non-ground rule r to domain value $d \in \mathrm{dom}(x)$. For (ii) non-existence of a model $N \subset M$, we require proper subsets. For this purpose we use $\mathrm{AtEq} := \{\mathrm{eq}_{p(X)} \mid p(X) \in \mathrm{AtPred}\} \cup \{\mathrm{nempty}\}$ to intuitively derive the equality of a predicate instantiation for the potential models M and N while considering empty candidates.

The overall idea consists of the following parts, which are encoded as our reduction R, which transforms a disjunctive, non-ground program Π into to an epistemic, ground program $\mathsf{R}(\Pi)$ consisting of $\mathcal{F}$ and the rules given in Figure 3.8. To generate all possible world view candidates, our targeted models M, we use Rules (R1) and (R2), where we

Guess Answer-Set Candidates

for every $h(X)\in\text{heads}(\Pi)$, $D\in\text{dom}(X)$, $a=h(D)$:
$a \leftarrow \textbf{not}\ \dot{a}$ (R1)

for every $h(X)\in\text{heads}(\Pi)$, $D\in\text{dom}(X)$, $a=h(D)$:
$\dot{a} \leftarrow \textbf{not}\ a$ (R2)

Guess Potentially $\subseteq$-Smaller Models

for every $h(X)\in\text{heads}(\Pi)$, $D\in\text{dom}(X)$, $a=h(D)$:
$a^{\mathsf{c}} \vee \dot{a}^{\mathsf{c}} \leftarrow a$ (R3)

Prevent Spuriously $\subseteq$-Smaller Models

for every $h(X)\in\text{heads}(\Pi)$, $D\in\text{dom}(X)$, $a=h(D)$:
$\text{eq}_a \leftarrow a, a^{\mathsf{c}}$ (R4)

for every $h(X)\in\text{heads}(\Pi)$, $D\in\text{dom}(X)$, $a=h(D)$:
$\text{eq}_a \leftarrow \neg a$ (R5)

for every $h(X)\in\text{heads}(\Pi)$, $D\in\text{dom}(X)$,$a=h(D)$
$\text{nempty} \leftarrow a$ (R6)

for every $h(X)\in\text{heads}(\Pi)$, $\{a_1,\ldots,a_n\}=\{h(D)|D\in\text{dom}(X)\}$, $1\leq i\leq n$:
$\leftarrow \text{nempty}, \text{eq}_{a_1}, \ldots, \text{eq}_{a_n}$ (R7)

Prevent Unsatisfied Rules and $\subseteq$-Smaller Models

$\leftarrow \textbf{not}\ \text{sat}$ (R8)

$\leftarrow \textbf{not}\ \neg\text{sat}^{\mathsf{c}}$ (R9)

Ensure Satisfiability of Non-Ground Rules: see Figure 3.7

Figure 3.8: Reduction $\mathsf{R}(\Pi)$ from a disjunctive, non-ground program Π to an epistemic, ground program.

use the epistemic negation to generate the mutually exclusive assignments (through atoms a and $\dot{a}$) at the outermost level (Elp). For (ii) we guess possible smaller models N throughout these candidates, achieved by Rule (R3), which results (for each world view candidate) in an answer set candidate for all possible subsets of the guessed world view candidates. Then, Rules (R4)–(R7) trivially derive the equality of the guessed head predicates ensuring that the guessed models N only contain proper subsets of models M, where we use Rules (R6) to consider and not remove the empty set candidate (for which no proper subset exists and the construction would result in removing it). As in earlier reductions, we ensure (i) satisfiability of models M by Rules (S1)–(S5), where we yield sat_r whenever there is an assignment of variables to domain values for a non-ground rule $r \in \Pi$. If such an atom can be derived for all non-ground rules $r \in \Pi$, we follow sat by Rule (S4), which is mandatory throughout the world view (cf. Rule (R8); an epistemic constraint to require atom sat to hold within a world view, i.e. for every answer set of the epistemic reduct sat holds). Rule (S5) applies *saturation*, causing the assignment of all domain values to every variable. This way any world view candidate M is removed if not complying with (i). In the same way, we ensure (ii) satisfiability for the potential models C (proper subsets of M), removing every answer set candidate (within

each world view candidate) that does not conform with (i). Notice that, in addition to Rules (S6)–(S8) and (S11), which are copying the behavior of Rules (S1)–(S3) and (S5), we use Rules (S9) to "satisfy" rules of Π that are removed by the construction of the GL reduct Π^M and instead of Rules (S4), we use Rules (S10) to consider a potentially empty model for which we do not need to check for satisfiability of (non-existing) proper subsets. Then, Rule (R9) removes any world view candidate M, where there is an answer set candidate deriving $\mathrm{sat}^{\mathsf{c}}$ such that we ensure (ii) non-existence of a model $C \subset M$ of Π^M. This is encoded through a constraint for the epistemic negation of the negated atom sat^c, which ensures that none of the answer sets of the epistemic reduct includes the atom sat^c. Intuitively, the only world view candidates remaining depict models of Π, where non of the answer set candidates contains $\mathrm{sat}^{\mathsf{c}}$, i.e. there is no proper subset C of the guessed model M (represented by the world view candidate) which represents a model of the GL Π^M.

Notice that, in comparison to the earlier presented reductions, the rules for guaranteeing foundedness of guessed head atoms and ordering are in our reduction R not required anymore. This is due to the fact that the reduction encodes subset-minimization (cf. Definition of an Answer Set, see Definition 2.4), which implicitly guarantees the other properties.

Example 3.6. *Consider the non-ground, disjunctive program* $\Pi_4 := \{a(X,Y), p(X,Y) \leftarrow b(X), c(Y,Z)\}$ *and facts* $\mathcal{F} := \{b(1).\ c(1,2).\}$. *Grounding the program as described above and shown in Figure 3.8, results in the ground program* $P_4 = \mathcal{F} \cup \mathsf{R}(\Pi_4)$ *of Figures 3.9 and 3.10. The world view candidates that are guessed by the resulting ground program are checked by the procedure described above, resulting in the two world views* $\{a(1,1), b(1), c(1,2)\}$ *and* $\{p(1,1), b(1), c(1,2)\}$ *(when restricted to the symbols of* Π_4*) which are equatable with the answer sets of the original non-ground program* Π_4.

Similar to the earlier reductions $\mathcal{R}$ and $\mathcal{R}''$ we state correctness of the reduction R in the following.

Theorem 3.5 (Correctness)**.** *Let* Π *be any disjunctive, non-ground program. Then, the grounding procedure* R *on* Π *is correct, i.e., the world views of* $\mathsf{R}(\Pi)$ *restricted to* $\mathrm{at}(\mathcal{G}(\Pi))$ *match with the answer sets of* $\mathcal{G}(\Pi)$. *Precisely, for every world view* W *of* $\mathsf{R}(\Pi)$ *there is exactly one answer set* M *of* $\mathcal{G}(\Pi)$, *such that* $M = \{a \mid a \in W \cap \mathrm{at}(\mathcal{G}(\Pi))\}$ *holds.*

Proof. $\Longrightarrow$: Let W be a WV of $\mathsf{R}(\Pi)$. Then, by Rules (R8), we require that $\mathrm{sat} \in W$. From this, we construct set $M := \{a \mid a \in W \cap \mathrm{at}(\mathcal{G}(\Pi))\}$. Then, it remains to show that (i) M is indeed a model of $\mathcal{G}(\Pi)$ and (ii) that M is a subset-minimal model of $\mathcal{G}(\Pi)^M$, i.e., there does not exist a model N of $\mathcal{G}(\Pi)^M$ with $N \subsetneq M$.

For establishing (i), we assume towards a contradiction that there is a rule $r \in \Pi$ with a concrete ground instantiation $r' \in \mathcal{G}(\{r\})$ such that M is not a model of r'. Then, we have that $B^+_{r'} \subseteq M$ and $(B^-_{r'} \cup H) \cap M = \emptyset$. As a consequence, by construction we have that

Ensure Satisfiabilty of a potential witness M

$\mathrm{gr}_X(1).\ \mathrm{gr}_Y(1) \vee \mathrm{gr}_Y(2).\ \mathrm{gr}_Z(1) \vee \mathrm{gr}_Z(2).$ (S1)

$\mathrm{sat}_r \leftarrow \mathrm{gr}_X(1), \neg b(1).$ (S2)

$\mathrm{sat}_r \leftarrow \mathrm{gr}_Y(1), \mathrm{gr}_Z(1), \neg c(1,1).\ \mathrm{sat}_r \leftarrow \mathrm{sat}_Y(1), \mathrm{gr}_Z(1), \neg c(1,2).$

$\mathrm{sat}_r \leftarrow \mathrm{gr}_Y(2), \mathrm{gr}_Z(1), \neg c(2,1).\ \mathrm{sat}_r \leftarrow \mathrm{gr}_Y(2), \mathrm{gr}_Z(1), \neg c(2,2).$

$\mathrm{sat}_r \leftarrow \mathrm{gr}_X(1), \mathrm{gr}_Y(1), a(1,1).$ (S3)

$\mathrm{sat}_r \leftarrow \mathrm{gr}_X(1), \mathrm{gr}_Y(2), a(1,2).$

$\mathrm{sat} \leftarrow \mathrm{sat}_r.$ (S4)

$\mathrm{gr}_X(1) \leftarrow \mathrm{sat}.\ \mathrm{gr}_Y(1) \leftarrow \mathrm{sat}.\ \mathrm{gr}_Y(2) \leftarrow \mathrm{sat}.$ (S5)

$\mathrm{gr}_Z(1) \leftarrow \mathrm{sat}.\ \mathrm{gr}_Z(2) \leftarrow \mathrm{sat}.$

Ensure Satisfiabilty of a potential counterwitness C

$\mathrm{gr}^{\mathsf{c}}_X(1).\ \mathrm{gr}^{\mathsf{c}}_Y(1) \vee \mathrm{gr}^{\mathsf{c}}_Y(2).\ \mathrm{gr}^{\mathsf{c}}_Z(1) \vee \mathrm{gr}^{\mathsf{c}}_Z(2).$ (S6)

$\mathrm{sat}^{\mathsf{c}}_r \leftarrow \mathrm{gr}^{\mathsf{c}}_X(1), \neg b^{\mathsf{c}}(1).$ (S7)

$\mathrm{sat}^{\mathsf{c}}_r \leftarrow \mathrm{gr}^{\mathsf{c}}_Y(1), \mathrm{gr}^{\mathsf{c}}_Z(1), \neg c^{\mathsf{c}}(1,1).\ \mathrm{sat}^{\mathsf{c}}_r \leftarrow \mathrm{sat}^{\mathsf{c}}_Y(1), \mathrm{gr}^{\mathsf{c}}_Z(1), \neg c^{\mathsf{c}}(1,2).$

$\mathrm{sat}^{\mathsf{c}}_r \leftarrow \mathrm{gr}^{\mathsf{c}}_Y(2), \mathrm{gr}^{\mathsf{c}}_Z(1), \neg c^{\mathsf{c}}(2,1).\ \mathrm{sat}^{\mathsf{c}}_r \leftarrow \mathrm{gr}^{\mathsf{c}}_Y(2), \mathrm{gr}^{\mathsf{c}}_Z(1), \neg c^{\mathsf{c}}(2,2).$

$\mathrm{sat}^{\mathsf{c}}_r \leftarrow \mathrm{gr}^{\mathsf{c}}_X(1), \mathrm{gr}^{\mathsf{c}}_Y(1), a^{\mathsf{c}}(1,1).$ (S8)

$\mathrm{sat}^{\mathsf{c}}_r \leftarrow \mathrm{gr}^{\mathsf{c}}_X(1), \mathrm{gr}^{\mathsf{c}}_Y(2), a^{\mathsf{c}}(1,2).$

$\mathrm{sat}^{\mathsf{c}} \leftarrow \mathrm{sat}^{\mathsf{c}}_r, \mathrm{nempty}.$ (S10)

$\mathrm{gr}^{\mathsf{c}}_X(1) \leftarrow \mathrm{sat}^{\mathsf{c}}.\ \mathrm{gr}^{\mathsf{c}}_Y(1) \leftarrow \mathrm{sat}^{\mathsf{c}}.\ \mathrm{gr}^{\mathsf{c}}_Y(2) \leftarrow \mathrm{sat}^{\mathsf{c}}.$ (S11)

$\mathrm{gr}^{\mathsf{c}}_Z(1) \leftarrow \mathrm{sat}^{\mathsf{c}}.\ \mathrm{gr}^{\mathsf{c}}_Z(2) \leftarrow \mathrm{sat}^{\mathsf{c}}.$

Figure 3.9: $\mathsf{S}(\Pi_4)$ with Π_4 from Example 3.6, guided by R of Figure 3.7.

there exists an answer set N of $\mathsf{R}(\Pi)^W$ with $\{\mathrm{sat}_x(d) \mid p(\langle x_1, \ldots, x_o, x, x_{o+1}, \ldots, x_u\rangle) \in B^+_r \cup B^-_r \cup H_r, p(\langle d_1, \ldots, d_o, d, d_{o+1}, \ldots, d_u\rangle) \in \mathrm{at}(r')\} \subseteq N$, $\mathrm{sat}_r \notin N$ by Rules (S2) and (S3). Consequently, by Rules (S4), $\mathrm{sat} \notin N$, which immediately contradicts our assumption that W is a WV due to Rules (R8); hence r' cannot exist.

For showing (ii), we assume towards a contradiction that there exists an interpretation $N \subsetneq M$ that is a $\subseteq$-smaller model of $\mathcal{G}(\Pi)^M$. From this we will now argue that therefore W cannot be a WV of $\mathsf{R}(\Pi)$. To this end, we construct an interpretation $M' := M \cup (\{\dot{h}(D) \mid h(X) \in \mathrm{heads}(\Pi), D \in \mathrm{dom}(X)\} \setminus M) \cup \{\mathrm{gr}_x(d), \mathrm{sat}_r, \mathrm{sat} \mid r \in \Pi, x \in \mathrm{var}(r), d \in \mathrm{dom}(x)\}$ as well as interpretations $N' := \{a^{\mathsf{c}} \mid a \in N\} \cup (\{\dot{h}^{\mathsf{c}}(D) \mid h(X) \in \mathrm{heads}(\Pi), D \in \mathrm{dom}(X)\} \setminus N) \cup \{\mathrm{gr}^{\mathsf{c}}_x(d), \mathrm{sat}^{\mathsf{c}}_r, \mathrm{sat}^{\mathsf{c}} \mid r \in \Pi, x \in \mathrm{var}(r), d \in \mathrm{dom}(x)\}$, and $E := \{\mathrm{nempty} \mid M \neq \emptyset\} \cup \{\mathrm{eq}_a \mid a \in (M \cap N) \cup \{b \mid \dot{b} \in M'\}\}$. Then, it remains to show that indeed $M' \cup N' \cup E$ is an answer set of $\mathcal{G}(\Pi)^W$, which, if W was a WV, is impossible by Rules (R9) and due to $\mathrm{sat}^{\mathsf{c}} \in N'$. Due to W and the construction of M',

Guess Answer-Set Candidates

$a(1,1) \leftarrow \textbf{not}\ \dot{a}(1,1).\ a(1,2) \leftarrow \textbf{not}\ \dot{a}(1,2).$ (R1)

$p(1,1) \leftarrow \textbf{not}\ \dot{p}(1,1).\ p(1,2) \leftarrow \textbf{not}\ \dot{p}(1,2).$

$\dot{a}(1,1) \leftarrow \textbf{not}\ a(1,1).\ \dot{a}(1,2) \leftarrow \textbf{not}\ a(1,2).$ (R2)

$\dot{p}(1,1) \leftarrow \textbf{not}\ p(1,1).\ \dot{p}(1,2) \leftarrow \textbf{not}\ p(1,2).$

Guess Potentially $\subseteq$-Smaller Models

$a^{\mathsf{c}}(1,1) \vee \dot{a}^{\mathsf{c}}(1,1) \leftarrow a(1,1).\ a^{\mathsf{c}}(1,2) \vee \dot{a}^{\mathsf{c}}(1,2) \leftarrow a(1,2).$ (R3)

$p^{\mathsf{c}}(1,1) \vee \dot{p}^{\mathsf{c}}(1,1) \leftarrow p(1,1).\ p^{\mathsf{c}}(1,2) \vee \dot{p}^{\mathsf{c}}(1,2) \leftarrow p(1,2).$

Prevent Spuriously $\subseteq$-Smaller Models

$\mathrm{eq}_{a(1,1)} \leftarrow a(1,1), \neg a^{\mathsf{c}}(1,1).\ \mathrm{eq}_{a(1,2)} \leftarrow a(1,2), \neg a^{\mathsf{c}}(1,2).$ (R4)

$\mathrm{eq}_{p(1,1)} \leftarrow p(1,1), \neg p^{\mathsf{c}}(1,1).\ \mathrm{eq}_{p(1,2)} \leftarrow p(1,2), \neg p^{\mathsf{c}}(1,2).$

$\mathrm{eq}_{a(1,1)} \leftarrow \neg a(1,1).\ \mathrm{eq}_{a(1,2)} \leftarrow \neg a(1,2).$ (R5)

$\mathrm{eq}_{p(1,1)} \leftarrow \neg p(1,1).\ \mathrm{eq}_{p(1,2)} \leftarrow \neg p(1,2).$

$\mathrm{nempty} \leftarrow a(1,1).\ \mathrm{nempty} \leftarrow a(1,2).$ (R6)

$\mathrm{nempty} \leftarrow p(1,1).\ \mathrm{nempty} \leftarrow p(1,2).$

$\leftarrow \mathrm{nempty}, \mathrm{eq}_{a(1,1)}, \mathrm{eq}_{a(1,2)}, \mathrm{eq}_{p(1,1)}, \mathrm{eq}_{p(1,2)}.$ (R7)

Prevent Unsatisfied Rules and $\subseteq$-Smaller Models

$\leftarrow \textbf{not}\ \mathrm{sat}.$ (R8)

$\leftarrow \textbf{not}\ \neg\mathrm{sat}^{\mathsf{c}}.$ (R9)

Figure 3.10: $\mathsf{R}(\Pi_4)$ with Π_4 from Example 3.6, guided by R of Figure 3.8.

Rules (R1) are facts or not present in $\mathsf{R}(\Pi)^W$.However, since M is a model of $\mathcal{G}(\Pi)$, we have that M' is by construction also a model of Rules (S1)–(S11). Then, by N, we similarly have that N' is model of Rules (S1)–(S5), as well. Further, $M' \cup N' \cup E$ is also a model of Rules (R3) (due to N') and of Rules (R4)–(R7) (due to E). Consequently, we have that $M' \cup N' \cup E$ is also a $\subseteq$-minimal model of $\mathcal{G}(\Pi)^W$; by construction and similar to (i), if there was $\subseteq$-smaller interpretation satisfying Rules (S6)–(S11) without containing some sat_r or $\mathrm{sat}^{\mathsf{c}}_{r'}$, then M or N would not be a model of $\mathcal{G}(\Pi)^W$, respectively.

Observe that due to incomparability of atoms $\dot{a}^{\mathsf{c}}$ and a^{c}, there cannot be a smaller model through Rules (R3) as well. We conclude that due to $M' \cup N' \cup E$ being an answer set of $\mathsf{R}(\Pi)^W$ despite $\mathrm{sat}^{\mathsf{c}} \in N'$, that therefore due to Rule (R9), W cannot be a WV of $\mathsf{R}(\Pi)$, contradicting our assumption.

$\Longleftarrow$: Let M be an answer set of $\mathcal{G}(\Pi)$. First, we construct $N := \{\neg a, \dot{a}^{\mathsf{c}}, \neg a^{\mathsf{c}}, \dot{a} \mid h(X) \in \mathrm{heads}(\Pi), D \in \mathrm{dom}(X), a = h(D), a \notin M\}$, which collects those head atoms that are not in M. From this we construct a set $I := M \cup N \cup \{\neg\mathrm{sat}^{\mathsf{c}}\} \cup \{\mathrm{sat}, \mathrm{sat}_r, \mathrm{gr}_x(d) \mid r \in$

$\Pi, x \in \mathrm{var}(r), d \in \mathrm{dom}(x)\} \cup \{\mathrm{nempty} \mid M \neq \emptyset\} \cup \{\mathrm{eq}_a \mid a \in \mathrm{at}(\mathcal{G}(\Pi)) \setminus M\}$, where we show that $I' \supseteq I$ is a WV of $\mathsf{R}(\Pi)$. Then, it remains to show that I can be extended by some subset S' of $S := \{\mathrm{sat}_r^{\mathsf{c}}, \neg\mathrm{sat}_r^{\mathsf{c}} \mid r \in \Pi\}$ to a WV of $\mathsf{P} = \mathsf{R}(\Pi)$, i.e., $I' := I \cup S'$ is compatible with $AS(\mathsf{P}^{I'})$.

Condition (1): $AS(\mathsf{P}^{I'}) \neq \emptyset$ is satisfied as Rules (R8) and (R9) resolve to false constraints by the construction of the epistemic reduct and Rules (R7) only constrain non-empty answer set candidates that yield identical sets of atoms a and a^{c}. However, these are only some candidates, as Rules (R3) suggest.

Condition (2): for each atom $a \in I$ and for any arbitrary $J \in AS(\mathsf{P}^{I'})$, we show $a \in J$ as follows. For every atom $a \in M$ the atom $a \in J$ holds (due to a fact in Rules (R1)) in every answer set of P^I by the construction of P^I. Similarly, for every atom $\dot{a} \in N$, P^I yields $\dot{a}$ in every answer set (by Rules (R1)). Notice that by construction of N there is no atom a such that $a \in M$ and $\dot{a} \in N$. Further, for $\mathrm{nempty} \in I$, we have $\mathrm{nempty} \in J$ since by construction there is some $a \in J$, required by some body of Rules (R6). For any atom of the form $\mathrm{eq}_a \in I$, similarly, by construction of I and Rules (R5), it holds that $\mathrm{eq}_a \in J$. Finally, for the remaining atoms $a \in \{\mathrm{sat}, \mathrm{sat}_r, \mathrm{gr}_x(d) \mid r \in \Pi, x \in \mathrm{var}(r), d \in \mathrm{dom}(x)\}$, we assume towards a contradiction that some $J \in AS(\mathsf{P}^{I'})$ does not contain a. Then, however, we have by construction of Rules (S4) and (S5) that there exists a rule $r \in \Pi$ such that $\mathrm{sat}_r \notin J$. In consequence, by Rules (S2) and (S3), we follow that M cannot be a model of $\mathcal{G}(\{r\})$, which contradicts the assumption that M is an answer set of $\mathcal{G}(\Pi)$.

Condition (3): for the negative literals in N, it is easy to see that by construction, none of these can appear in any answer sets $J \in AS(\mathsf{P}^{I'})$. Further, for the negative literal $\neg\mathrm{sat}^{\mathsf{c}} \in I$, we also have that for every $J \in AS(\mathsf{P}^{I'})$, $\mathrm{sat}^{\mathsf{c}} \notin J$, which is satisfied as follows. Assume towards a contradiction that $\mathrm{sat}^{\mathsf{c}} \in J$. Then, by construction, we have that $M \neq \emptyset$. Consequently, since J has to be a model of Rules (R7), we can construct a set $N' \subsetneq M$ that is a model of $\mathcal{G}(\Pi)^M$ as follows: $N' := \{a \mid a^{\mathsf{c}} \in J\}$. Since J is an answer set of P^I there cannot be a smaller model $J' \subsetneq J$ with $\mathrm{sat}^{\mathsf{c}} \notin J'$. Therefore, N' is indeed a model of $\mathcal{G}(\Pi)^M$, since, independent of the instantiation constructed by Rules (S1), every rule $r \in \Pi$ is satisfied by N' due to $\mathrm{sat}^{\mathsf{c}} \in J$ and construction of Rules (S7)–(S9). This, however, contradicts that M is an answer set.

Condition (4): For every atom $b \in \mathrm{at}(\mathsf{P})$ with $\{b, \neg b\} \cap I = \emptyset$, there are $J, J' \in AS(\mathsf{P}^{I'})$ with $b \in J$ and $b \notin J'$. For each atom of the form $b = \dot{a}$, one can easily find both J, J' by Rules (R3); similarly, for $b = \mathrm{eq}_a$ we can find J,J' according to Rules (R4) by first setting either $a^{\mathsf{c}} \in J$ or not. Further, by Rules (S6), for atoms of the form $b = \mathrm{gr}_x(d)^{\mathsf{c}}$, one can also easily construct both J, J' accordingly.

Finally, observe that the remaining literals $\ell \in S$ either belong to Condition (1), (2), or (3), where we require either atom $\mathrm{at}(\ell) \in S'$, literal $\neg\,\mathrm{at}(\ell) \in S'$, or neither, respectively. This concludes that $I' = I \cup S'$ is a WV of $\mathsf{R}(\Pi)$. □

Interestingly, the evaluation of the runtime of reduction R shows that the upper bound is lower than for any other reduction mentioned before. This is due to the fact that

reduction R encodes subset minimization and therefore does neither require checking foundedness (cf. Fig. 3.1) nor orderings of derivation (cf. Fig. 3.5), which depict the most effort of the other reductions.

Theorem 3.6 (Runtime)**.** *Let* Π *be any disjunctive, non-ground program, where every predicate has arity at most* a*. Then, the grounding procedure* R *on* Π *is polynomial, i.e., runs in time* $\mathcal{O}(\|\Pi\| \cdot |\mathrm{dom}(\Pi)|^a)$.

Proof. The reduction R constructs $\mathcal{O}(\|\Pi\| \cdot |\mathrm{dom}(\Pi)|^a)$ many Rules (R1), (R2) and (R3) of constant size for guessing candidates. For preventing spuriously smaller models $\mathcal{O}(\|\Pi\| \cdot |\mathrm{dom}(\Pi)|^a)$ many Rules (R4), (R5) and (R6) of constant size, as well as one Rule (R7) of size $\mathcal{O}(|\Pi|)$ are constructed. For the sub procedure S, we require: $\mathcal{O}(\|\Pi\| \cdot a)$ many Rules (S1) and (S6) of size $|\mathrm{dom}(\Pi)|$, $\mathcal{O}(\|\Pi\| \cdot |\mathrm{dom}(\Pi)|^a)$ many Rules (S2). Then, (S3), (S7), (S8) and (S9) of size $\mathcal{O}(a)$, one of each Rule (S4) and (S10) of size $\mathcal{O}(\|\Pi\|)$, as well as $\mathcal{O}(\|\Pi\| \cdot |\mathrm{dom}(\Pi)|)$ many Rules (S5) and (S11) of constant size. Further, the reduction R constructs one Rule (R8) and (R9) of constant size. □

CHAPTER 4

Implementation & Experiments

In this chapter we turn our focus to practically applying body-decoupled grounding introduced in earlier chapters. To this end, we implement a prototype system which follows the idea of our presented reduction to achieve grounding of non-ground programs via reduction. The system and technical details are presented in Section 4.1. Then, experiments and evaluations of our approach in practice are conducted in Sections 4.2 and 4.3.

4.1 System Overview

To show our approach in practical environments we implemented an open-source, prototype software tool called NEWGROUND[1]. Accordingly, the system NEWGROUND implements our reduction $\mathcal{R}'$ for non-ground, tight programs to realize body-decoupled grounding via search as described in the earlier chapter.

To this end, we designed NEWGROUND as reusable module that translates non-ground input programs using our reductions into ground, disjunctive programs. This way the system stays independent of any potentially used solver, i.e. the user stays in control of which grounder, solver etc. is used afterwards, making NEWGROUND thoroughly universally applicable (even for future solvers). For this reason we settled for a command line tool, which is not only basically standard for ASP related software, but also allows for fast piping of inputs and outputs as well as flawlessly cooperating with Python3, which was used for the implementation. Further, we utilize the latest CLINGO Python API for efficiently parsing inputs profiting from clingo's longstanding development of efficient ASP software.

[1] Our system including supplemental material of this work is publicly available at `https://github.com/viktorbesin/newground`.

As implied, the system NEWGROUND implements the optimized reduction $\mathcal{R}'$ for non-ground, tight programs featuring the improved rule set for checking foundedness which utilizes variables independencies, as highlighted in Figure 3.3. Further, as we still believe in the strengths of traditional, sophisticated grounding systems, NEWGROUND also supports partial reducability (cf. Corollary 3.1). This allows users to select program parts that shall be grounded using our reduction and others that are traditionally grounded, which results in even more freedom in usage for users.

4.1.1 Technical Details

While the system NEWGROUND is meant as a prototype for showing proof of work and to execute preliminary experiments, we still opted for a performance oriented approach. Therefore, NEWGROUND relies on the latest clingo 5.5 Python API[2], which, driven by years long of development, not only simplifies logic program parsing, but parses very efficiently into well-documented data structures. To this end, we use clingos `clingo.application` package, which enables the interaction with the typical parsing, grounding and solving process of CLINGO. The control flow of NEWGROUND can be seen in Figure 4.1 and is briefly explained in the following.

First, NEWGROUND uses the mentioned API to parse the given non-ground logic program into its data structures and objects. To differentiate between parts being subject to the reduction and others being left out for traditional grounding, NEWGROUND distinguishes the input, as supported by clingos parsing feature, by different subprograms. Next, NEWGROUND uses the generated Control object of the `clingo.control` package to read out facts and to be untouched program parts and appends them to the desired output. Depending on whether a reduction is requested (by the appropriate subprogram of the input, cf. 4.1.2), NEWGROUND starts its process of reducing non-ground rules. To this end, we make use of clingos `clingo.ast` package, which enables the traversal of non-ground logic programs in terms of *abstract syntax trees* by using its Transformer class. The reduction process itself then contains two full traversals of the input program. In the first traversal, NEWGROUND picks up predicate domains and facts of the whole program, which are needed for generating the desired ground rules. The second then conducts the actual translation of non-ground rules by picking one after another and generating the ground rules for each of the earlier stored domain values, as presented in Chapter 3. At this point, we let NEWGROUND pre-check arithmetic conditions of rule bodies to minimize unnecessary output that might be generated through domain values. When all rules are translated, the output is printed.

Notice that, at this point, the output might not be fully grounded as there might be a subprogram in the input that is not being part of the reduction. But as suggested earlier, partial reducability allows for *partial grounding* of this type, where the rest of the non-ground program can be grounded by any other grounder. Similarly, potentially any solver can be used on the output printed by NEWGROUND. This way the tool can further

[2]More details can be found at `https://potassco.org/clingo/python-api/5.5/`.

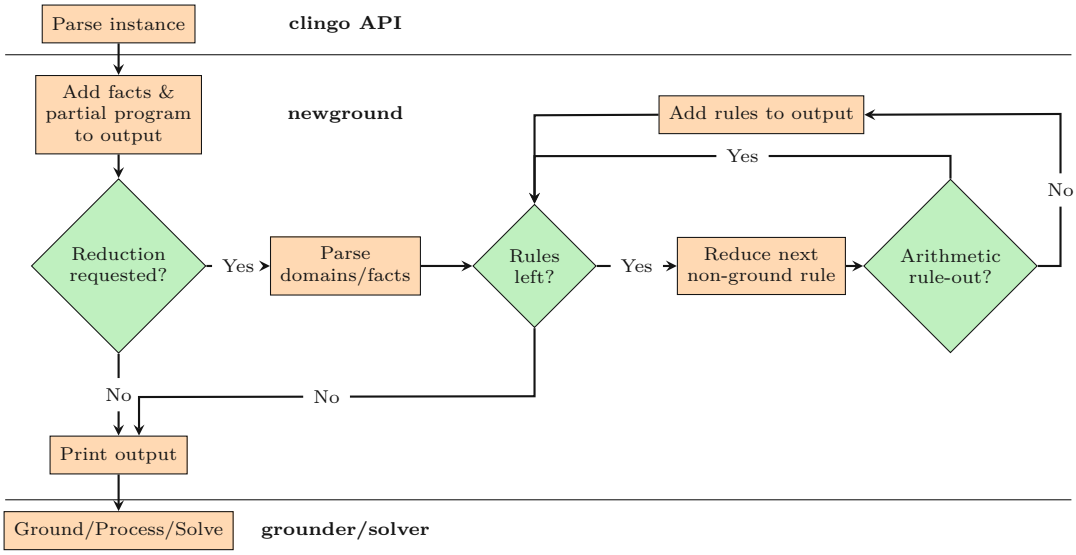

Figure 4.1: Flowchart that shows how NEWGROUND parses inputs using the CLINGO API and outputs partially ground programs.

benefit from and be used with any future solver. Indeed, NEWGROUND features a direct grounding after its implemented reduction. By using the `clingox` package we enable users to fully ground any remaining non-ground program parts.

4.1.2 Usage

The tool NEWGROUND can be used similar to CLINGO directly from the command line. For running NEWGROUND, CLINGO and the following Python3 packages are required to be installed: `clingo`, `clingox`, and `future-fstrings`.

The input format for NEWGROUND is equivalent to clingos input format. Based on the principle of partial reducability (cf. 3.1), inputs can be divided into parts that shall be part of the reduction by using subprograms. For this reason users may use `#program rules.` for (non-ground) program parts that shall be reduced by NEWGROUND. Additionally, the sub-program `#program insts.` can be used for instantiating the program with facts.

Without explicit domains given the reduction uses the complete set of terms to fill all variables in the grounding process. This process can be simplified by giving a domain for each variable using the associated predicate `_dom_()`, e.g. `_dom_X(1..5).`, or with more generic rules of the form `_dom_X(X) :- a(X,_).`, in the instatiating-part of the program. This information is then processed automatically and considered in the reduction.

The command line usage is shown in Listing 4.1. Notice that, besides the option `--ground` for fully grounding the output, NEWGROUND features options to ground

any choice rules part of the reduction as well as the suppressing of any show-rules for compatibility with solvers not implementing these types of rules.

```
$ usage: newground [files]

positional arguments:
file

optional arguments:
-h, --help      show this help message and exit
--no-show       Do not print #show-statements to avoid
    compatibility issues.
--ground-guess  Additionally ground guesses which results in
    grounded output.
--ground        Output program fully grounded.
```

Listing 4.1: Usage details of NEWGROUND.

4.2 Benchmarks

In order to evaluate the performance of NEWGROUND, we design a series of benchmarks. Clearly, we cannot beat established and highly optimized grounders in all imaginable scenarios and instances. Instead, the goal of our prototype and these benchmark settings is to visualize and discuss the potential and strengths of our approach. Further, we point out use cases, where body-decoupled grounding might be preferable with its advantages over traditional approaches. Indeed, body-decoupled grounding can be incorporated into every grounder, e.g. as part of a portfolio. These benchmark settings are used to discuss potential ways to do so.

4.2.1 Benchmark Scenarios

In order to test NEWGROUND, the following (directed) graph *scenarios* are considered.

S1 (coloring): Compute *edge colorings* over three colors in directed graphs such that for each node no incoming or outgoing edges have the same color. See Listing 4.2.

S2 (paths): Find *reachable paths* between source and destination nodes, where each node admits only one outgoing edge. See Listing 4.3.

S3 (clique): Obtain directed subgraphs containing so-called *cliques*, i.e. fully connected subgraphs of size at least three. See Listing 4.4.

S4 (nprc): Compute non-partition-removal colorings: remove one vertex such that the transitive closure of the original and the resulting graph are equal on the remaining nodes and that the resulting graph is 3-colorable. Encoding is taken from [WTF20]. See Listing 4.5.

S5 (stable marriage): Obtain so-called *stable marriages*. Encoding is taken from the ASP competition 2014. See Listing 4.6.

The selection of scenarios is justified as follows. With Scenario S1 (coloring) we aim at providing a basic coloring problem. While the decision of Scenario S2 (paths) is in P, the problem can be extended to counting: counting such paths is as hard as every #P problem [Val79]. Deciding Scenario S3 (clique) is in polynomial-time computable, but ready to be used as part of more expressive problems. Finally, we consider Scenarios S4 (nprc) and S5 (stable marriage) in order to cover known ASP scenarios.

Partial Reductions for newground As body-decoupled grounding is meant as a relief for those rules that hold a crucial role in terms of grounding size (grounding bottleneck), we chose to apply our reduction through NEWGROUND for the following rules.

S1 (coloring): on the rules prohibiting identical edge colors

S2 (paths): on the rule restricting to 1 chosen outgoing edge per node

S3 (clique): on the rule checking for a clique

S4 (nprc): on a rule that ensures non-partition (over reachability)

S5 (stable marriage): on the rules preventing polygamy

In the encodings, the respective rules below "`#program rules.`" are the ones subject to reduction and NEWGROUND grounds all other rules via GRINGO or IDLV (NEWGROUND*). For all other benchmarked vanilla solvers, the lines indicating subprograms are removed before those solvers are invoked.

```
1  % guess coloring
2  { g(X,Y); b(X,Y); r(X,Y) } :- edge(X,Y).

4  % only one color
5  :- g(X,Y), b(X,Y).
6  :- g(X,Y), r(X,Y).
7  :- r(X,Y), b(X,Y).

9  #program rules.
10 % not coloring 2 outgoing edges the same
11 :- g(X,Y), g(X,Z), Y < Z.
12 :- b(X,Y), b(X,Z), Y < Z.
13 :- r(X,Y), r(X,Z), Y < Z.

15 % not coloring 2 ingoing edges the same
16 :- g(Y,X), g(Z,X), Y < Z.
17 :- r(Y,X), r(Z,X), Y < Z.
18 :- b(Y,X), b(Z,X), Y < Z.
```

Listing 4.2: Encoding for Benchmark Scenario S1.

```
1  % start is reachable
2  r(X) :- X=#min{ Y: edge(Y,_); Y: edge(_,Y) }.
3  % destination has to be reachable
4  :- not r(X), X=#max{ Y: edge(Y,_); Y: edge(_,Y) }.

6  % guess used edges / path
7  { f(X,Y) } :- edge(X,Y).

9  % reachability
10 r(A) :- r(B), f(B,A).

12 #program rules.
13 % not more than 2 outgoing rules
14 :- f(B,A), f(B,C), A != C.
```

Listing 4.3: Encoding for Benchmark Scenario S2.

```
1  % guess used edge
2  { f(X,Y) } :- edge(X,Y).

4  #program rules.
5  % has to contain at least 3-clique
6  c :- f(A,B), f(A,C), f(B,C), A != B, B != C, A != C.
7  :- not c.
```

Listing 4.4: Encoding for Benchmark Scenario S3.

```
1  vertex(X) :- edge(X,_).
2  vertex(Y) :- edge(_,Y).
3  keep(X) :- vertex(X), not delete(X).
4  delete(X) :- vertex(X), not keep(X).
5  :- delete(X), vertex(Y), not keep(Y), X != Y.
6  kept_edge(V1, V2) :- keep(V1), keep(V2), edge(V1, V2).
7  reachable(X, Y) :- kept_edge(X, Y).
8  blue(N) :- keep(N), not red(N), not green(N).
9  red(N) :- keep(N), not blue(N), not green(N).
10 green(N) :- keep(N), not red(N), not blue(N).
11 :- kept_edge(N1,N2), blue(N1), blue(N2).
12 :- kept_edge(N1,N2), red(N1), red(N2).
13 :- kept_edge(N1,N2), green(N1), green(N2).
14 reachable(X, Z) :- delete(D), edge(X, D), reachable(X, Y),
       reachable(Y, Z).

16 #program rules.
17 :- delete(D), edge(V1, D), edge(D, V2), not reachable(V1, V2).
```

Listing 4.5: Encoding for Benchmark Scenario S4.

```
% guess matching
match(M,W) :- manAssignsScore(M,_,_), womanAssignsScore(W,_,_),
    not nonMatch(M,W).
nonMatch(M,W) :- manAssignsScore(M,_,_), womanAssignsScore(W,_,_)
    , not match(M,W).

% no singles
jailed(M) :- match(M,_).
:- manAssignsScore(M,_,_), not jailed(M).

% strong stability condition
:- match(M,W1), manAssignsScore(M,W,Smw), W1 <> W,
    manAssignsScore(M,W1,Smw1), Smw > Smw1, match(M1,W),
    womanAssignsScore(W,M,Swm), womanAssignsScore(W,M1,Swm1), Swm
    >= Swm1.

#program rules.
% no polygamy
:- match(M1,W), match(M,W), M <> M1.
:- match(M,W), match(M,W1), W <> W1.
```

Listing 4.6: Encoding for Benchmark Scenario S5.

Based on these scenarios, we study corresponding hypothesis that shall be verified in this section.

H1: In contrast to traditional grounding, body-decoupled grounding of NEWGROUND suffers less from increased *instance density* and *instance size*.

H2: Body-decoupled grounding can massively reduce *grounding sizes* and *grounding times* of large instances.

H3: Body-decoupled grounding can *improve overall and solving performance* on crafted and application instances.

H4: The idea of body-decoupled grounding, where suitable, efficiently interoperates with other approaches.

4.2.2 Benchmark Instances

For answering the hypotheses, we use crafted (random) and applicable ASP competition instances where applicable. Note that *competition instances are not particularly designed* to run into grounding bottlenecks. Therefore, we randomly generate instances, (directed) graphs of different size ranging from 100 to 1500 vertices with an edge probability (density) from 0.1 to 1.0, for S1–S4. These are particularly useful to answer Hypotheses H1–H3. For S5, we took competition instances (as well as crafted ones), which aid in analyzing Hypotheses H2–H4.

4.2.3 Compared Tools

For the evaluation of our tool we only study the performance of the following *exact (full) grounders*, i.e. we do not consider any other modern approaches, e.g. lazy grounding or Asp module theory.

- gringo: version 5.5.1
- idlv: version 1.1.6
- newground: where Body-decoupled grounding is applied on certain (manually fixed, see Section 4.2.1) non-ground rules of the respective programs that potentially cause grounding overhead. The remaining part is grounded with gringo.
- newground*: similar to newground with idlv being used instead of gringo.

While running these benchmarks we measure *grounding sizes* and *grounding times* of the mentioned grounders. Further, we measure *solving capabilities* of the resulting groundings for comparing solving. For full intercomparability, grounding size measures the size of the corresponding text output without writing it to the persistent storage (no disk or I/O operations). Anyway, relative orders of magnitude of measured grounding sizes unambiguously stand and are independent of format optimizations (which could be applied to any grounder).

For comparing the *overall performance* of the groundings, we use solver clingo version 5.5.1 with options `-q --stats=2` to compute one answer set without any excessive output. For solving the groundings of newground we add the option `--project` to ensure answer sets are over the same atoms. We *limit* the main memory (RAM) of the cluster to 16GB and overall runtimes (grounding *& solving*) to 1800s. For all resulting plots we *cut-off* grounding sizes beyond 10GB (or 30GB where reasonable).

4.2.4 Benchmark Platform

All of our benchmarks were conducted on a cluster consisting of 12 nodes. Each node of the cluster is equipped with two Intel Xeon E5-2650 CPUs and each of these 12 physical cores runs at 2.2 GHz clock speed that has access to 256 GB shared RAM. Results are gathered on a system running Ubuntu 16.04.1 LTS OS that is powered on kernel 4.4.0-139. We disabled hyperthreading and used Python version 3.7.6.

4.3 Results

In the following we discuss the results of the performed benchmarks, as stated above. To this end, we evaluate the results in terms of grounding scalability, grounding performance and overall performance to verify our hypothesis H1–H4. Further, where appropriate, we compare to the better performing contestant (grounding profiles) and similar, we limit the results to the better performing alternative of newground (grounding times).

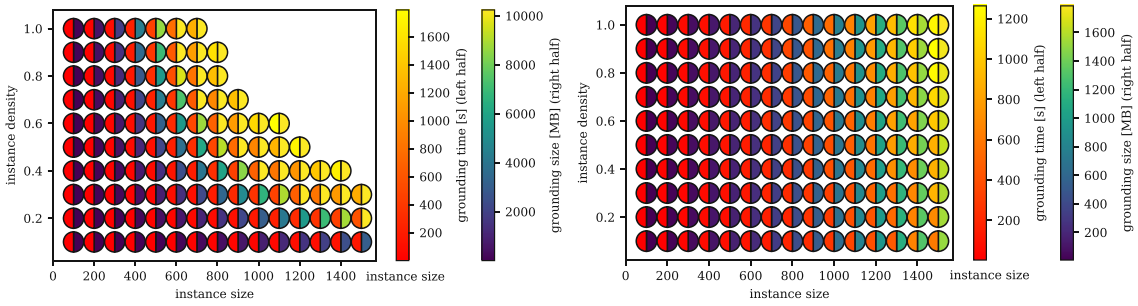

Figure 4.2: (Left): Grounding profile for GRINGO of S1 (coloring). (Right): Grounding profile for NEWGROUND of S1. The x-axis refers to the instance size and the y-axis indicates the density. A circle indicates that an instance was grounded below $< 1800s$, where the left half depicts grounding time and the right half depict grounding size. Mind the different scales.

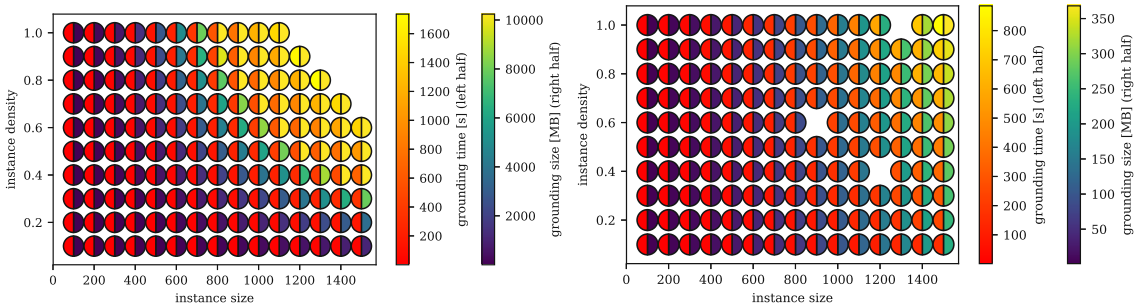

Figure 4.3: Similar to Figure 4.2. (Left): Grounding profile for GRINGO of S2 (paths). (Right): Grounding profile for NEWGROUND of S2.

Grounding Scalability For studying and comparing the groundings yielded by the different tools, we use crafted instances for Scenarios S1–S4 to generate a grounding profile for each tool. Figure 4.2 to Figure 4.5 presents these grounding profiles for NEWGROUND and its better performing contestant, where grounding times and sizes are shown depending on the instances size (x-axis) and instance density (y-axis). Circles indicate a grounding below 1800s where the left half depicts the grounding time and the right half the grounding size. Notice that different scales are used to allow for a more distinct value read-off throughout the plots.

Throughout the plots it can be seen that, compared to GRINGO (and IDLV), NEWGROUND grounds larger and denser instances faster. Further, one can clearly identify the reduction of grounding sizes of up to $\frac{1}{50}$ (cf. Fig. 4.2.1). Interestingly, when looking at fixed instance sizes (columns) for NEWGROUND (Fig. 4.2 to 4.5, (right)), the circles show very similar results, i.e. the colors do not vary much. In comparison, this does not hold for GRINGO (and IDLV), which suggests that NEWGROUND does in fact not suffer as much from increasing instances density compared to GRINGO and IDLV (Fig. 4.2 to 4.5, (left)).

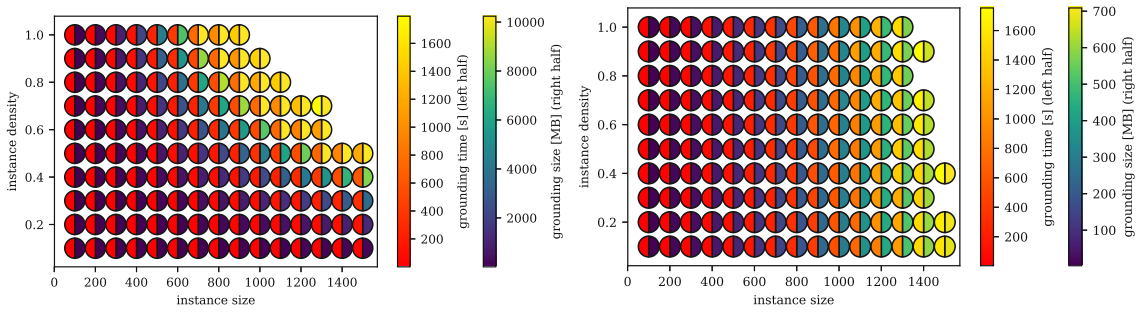

Figure 4.4: Similar to Figure 4.2. (Left): Grounding profile for GRINGO of S3 (clique). (Right): Grounding profile for NEWGROUND of S3.

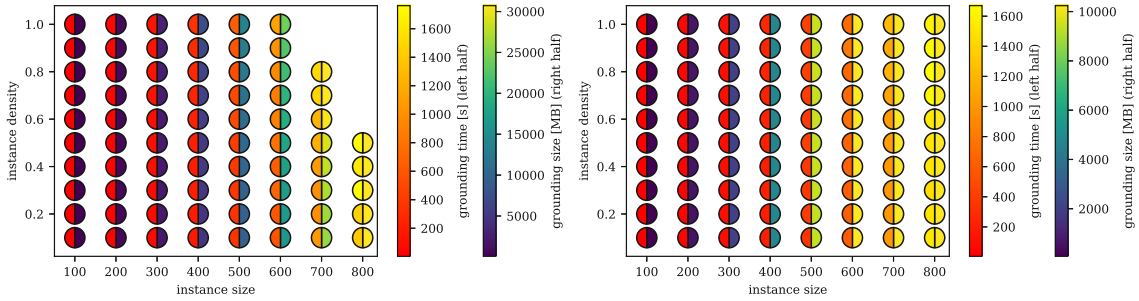

Figure 4.5: Similar to Figure 4.2. (Left): Grounding profile for IDLV of S4 (nprc). (Right): Grounding profile for NEWGROUND of S4.

In a similar matter, we identify that the rows are also more similar, which is why we overall can confirm hypothesis H1.

Grounding Performance For evaluating the actual groundings of the tools, we again use our crafted instances for scenarios S1–S4 and generate cactus for each tool cactus and scatter plots to further assess the total value distribution. Notice that the plots show either NEWGROUND or NEWGROUND* according to which was better performing.

Figure 4.6 (left) shows the cactus plot of grounding times for scenarios S1–S4 for all grounders. It can be identified that NEWGROUND outperforms (in terms of grounding time) in these scenarios with GRINGO performing second-best, expect for S4 (nprc). We believe that this is most likely due to the fact that S4 allows to utilize treewidth-based methods (as IDLV uses them). Consequently, the plots show NEWGROUND* for S4, demonstrating that NEWGROUND works well together with IDLV. Similar, Figure 4.7 (left) shows grounding times for S5 (stable marriage), where the trend that NEWGROUND performs well continues.

The scatter plots in Figure 4.6 (right) and Figure 4.7 (right) for grounding times depict similar results. Again, we can identify that NEWGROUND improves groundings times in most cases as we see more blue/green dots above the diagonal. Among those below

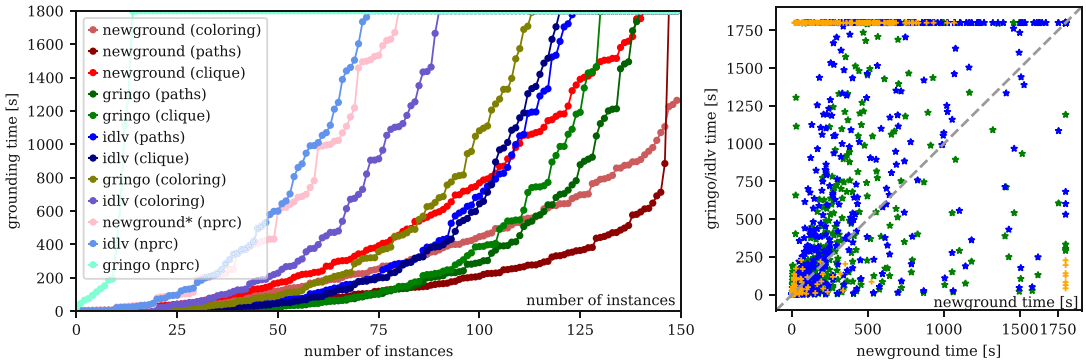

Figure 4.6: (Left): Cactus plot of grounding time over Scenarios S1–S4 for NEWGROUND (red color tones), GRINGO (green tones), and IDLV (blue tones). The x-axis shows the number of instances and the y-axis the runtime in seconds, sorted in ascending order for each solver individually. The legend is sorted from best to worst. (Right): Scatter plot of grounding time over Scenarios S1–S4 of NEWGROUND (x-axis) compared to GRINGO and IDLV (y-axis). Overall times (grounding *and solving time*) of solved instances are highlighted in orange.

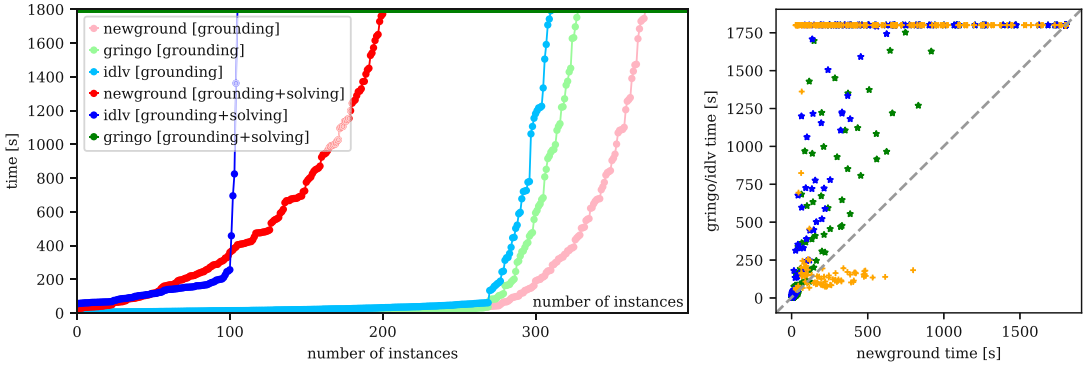

Figure 4.7: (Left): Cactus plot of grounding time as well as overall performance for Scenario S5 (stable marriage). (Right): Scatter plot of grounding time over S5.

the diagonal (where GRINGO and IDLV are faster), most dots are below 250s, suggesting a potential for a portfolio that uses GRINGO or IDLV for a limited time of 250s and switches to NEWGROUND if unsuccessful. Especially, since none (for S5) of the instances are actually solved beyond that time (orange dots below the diagonal).

The scatter plots of Figure 4.8 give some indication regarding grounding sizes. As most of the dots are above the diagonal, it can be seen that NEWGROUND massively reduces groundings sizes. Moreover, there are instances where GRINGO and IDLV output groundings beyond 30GB which are still *solved* by NEWGROUND.

From these evaluations we can confirm hypothesis H2.

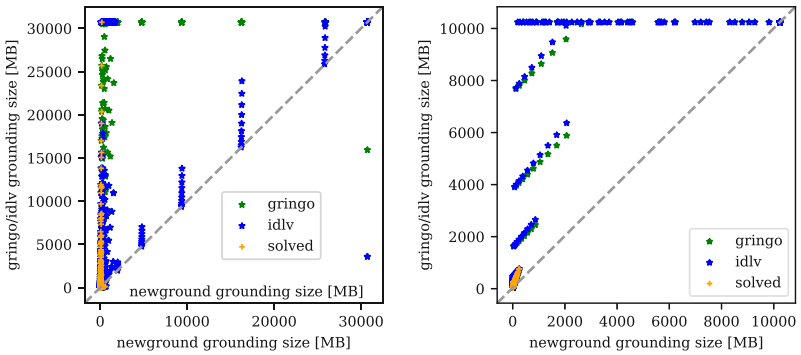

Figure 4.8: (Left): Scatter plot of grounding size over Scenarios S1–S4 of NEWGROUND (x-axis) compared to both GRINGO (blue) and IDLV (green) on the y-axis. (Right): Scatter plot of grounding size over Scenario S5. Those instances that could be solved are highlighted in orange.

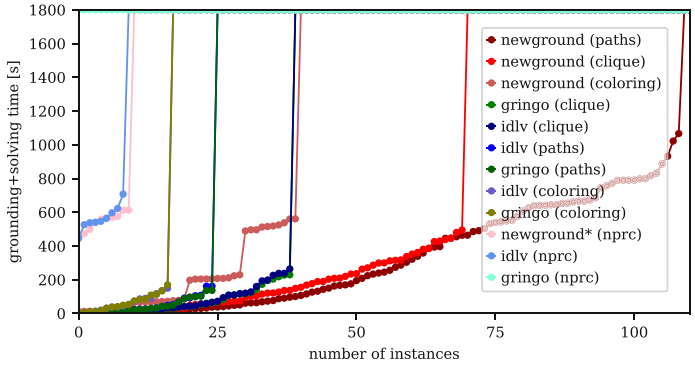

Figure 4.9: Corresponding cactus plot of overall (grounding and solving) time over Scenarios S1-S4 (cf. grounding performance of Figure 4.6 (Left)).

Overall Performance For the evaluation of solvability of the yielded groundings we refer to Figure 4.9 and Figure 4.7 (left), which show the overall runtime over Scenarios S1–S5. Besides the fact that NEWGROUND still performs better than its contestants, one can identify an immense difference between grounding and solving performance when comparing with Figure 4.6. Interestingly, while it may seem unfair to analyze combined grounding and solving time (since NEWGROUND grounds faster), the comparison with the cactus plot for grounding time only reveals that many instances that are grounded within short time by GRINGO or IDLV, but not solved within the remaining 1600s, e.g. many instances of S2 and S3 are grounded by GRINGO within 200s (cf. Fig. 4.6 (left)), but only a fraction of those are solved as Figure 4.6 shows. This is especially visible when combining grounding and overall time in a plot, as in Figure 4.7 (left) for Scenario S5. While IDLV solves a good amount of grounded instances before dropping out, NEWGROUND can withstand a higher number of instances with a more gentle increase of overall time.

Overall, we see H3 confirmed. Further, throughout the benchmarks we see NEWGROUND operating well with GRINGO and IDLV, which is why we see confirmed H4 as well.

Table 4.1 shows detailed results over Scenarios S1–S5, which further prove the overall performance of NEWGROUND. In most scenarios NEWGROUND shows higher number of grounded and solved instances, while keeping grounding times and grounding sizes lower than the compared tools GRINGO and IDLV.

	grounded instances					grounded+solved instances			
solver (scenario)	∑	grounding time[h]	grounding[GB]	max(grounding[MB])	max(gringo size[MB])	∑	time[h]	fastest	unique
newground (S1:coloring)	**150**	**14.5**	**87.0**	**1785.69**	**102400.0**	**40**/150	**57.27**	**32**	**23**
gringo (S1:coloring)	114	29.44	4346.11	34612.96	34612.96	17/150	66.78	5	0
idlv (S1:coloring)	91	39.37	6165.15	13070.43	13070.43	17/150	66.78	3	0
newground (S2:paths)	**147**	**8.6**	**315.17**	**368.1**	**102400.0**	**109**/150	**30.55**	**101**	**84**
gringo (S2:paths)	140	15.54	1783.91	34431.63	34431.63	25/150	62.8	8	0
idlv (S2:paths)	124	24.61	2998.41	19011.75	19011.75	25/150	62.82	0	0
newground (S3:clique)	**141**	25.02	**931.96**	**708.82**	**102400.0**	**70**/150	**43.12**	**52**	**38**
gringo (S3:clique)	131	**18.73**	2397.33	27465.99	27465.99	39/150	56.24	16	0
idlv (S3:clique)	120	24.84	3280.83	17836.62	17836.62	39/150	56.29	9	0
newground * (S4:nprc)	**80**	**46.21**	**7953.04**	**38732.34**	**102400.0**	**10**/150	**71.51**	**6**	**1**
gringo (S4:nprc)	15	69.27	13187.82	57268.14	57268.14	0/150	75.0	0	0
idlv (S4:nprc)	73	47.96	8505.09	43224.31	**102400.0**	9/150	71.91	4	0
newground (S5,competition)	**50**	0.1	**1.12**	**32.49**	**84.26**	**49**/50	**1.47**	**45**	**1**
gringo (S5,competition)	**50**	**0.05**	2.88	84.26	**84.26**	0/50	25.0	0	0
idlv (S5,competition)	**50**	0.14	3.21	94.19	**84.26**	48/50	2.56	4	0
newground (S5,crafted)	**322**	**29.76**	**3319.58**	23596.12	**102400.0**	**152**/350	**125.94**	**100**	**95**
gringo (S5,crafted)	278	47.27	8096.1	45380.83	45380.83	0/350	175.0	0	0
idlv (S5,crafted)	261	54.39	9302.68	**21822.21**	21593.23	57/350	149.28	52	0
newground	**890**	**124.18**	**12607.87**	**38732.34**	**102400.0**	**430**/1000	**329.86**	**336**	**242**
gringo	728	180.3	30314.16	57268.14	57268.14	81/1000	460.82	29	0
idlv	719	191.31	30255.38	43224.31	**102400.0**	195/1000	409.63	72	0

Table 4.1: Detailed results over Scenarios S1–S5, where best performance values are highlighted. The block "grounded instances" shows the number of grounded instances (∑), the total ground time in hours ("grounding time[h]", timeouts count as 1800s), the total grounding size in GB ("ground[GB]", non-groundable instances count as 100GB), the largest grounding size encountered ("max(grounding[MB])"), and the largest groundable instance given in the corresponding gringo grounding size ("max(gringo size[MB])"). The block "grounded+solved instances" refers to overall performance values, where "∑" shows the number of solved instances (in comparison to the number of total instances), "time[h]" gives the overall solving time in hours (timeouts: 1800s), "fastest" refers to the number of instances solved first, and "unique" gives the number of uniquely solved instances.

CHAPTER 5

Discussion

Finally, in this chapter we discuss and summarize the results of this thesis by putting them into perspective of related work. Further, by interpreting the results and consequences, we give rise to future work.

5.1 Summary

This thesis deals with the question if non-ground logic programs can be efficiently grounded to ground programs by a translation procedure while circumventing the Asp grounding bottleneck. Relying on the complexity studies for non-ground, normal and tight programs under bounded predicate arity, we first introduce a reduction to ground, disjunctive programs as a potential alternative to traditional grounding. To do so, the translation replaces non-ground rules by wisely encoding guessing of candidates, saturation foundedness (and orderings), which allows the decoupling of rules bodies. By decoupling dependencies between body predicates the instantiation can be limited to the linear size of the ground atoms. In similar fashion, we introduce a reduction for non-ground, disjunctive logic programs. However, due to the increased complexity, we make use of epistemic logic programs. While the core idea of the translation is similar, the encoding involves subset minimization by guessing epistemic atoms and checking saturation for both, potential model and counter witness. Nevertheless, body-decoupling is guaranteed through the encoding. While both reductions yield groundings that are linear in the size of the domain under the setting of bounded predicate arity, without this restriction we still achieve groundings exponential in the predicate arity.

In terms of runtime, we have shown polynomial run times throughout all introduced reduction procedures $\mathcal{R}$, $\mathcal{R}''$ and R. We do not expect a significant runtime improvement in the worst case. However, we have seen that orderings tremendously increase the run time of procedure $\mathcal{R}''$ compared to $\mathcal{R}$ and R, where they are either not needed or can be omitted.

The prototype NEWGROUND[1] implements our optimized reduction procedure $\mathcal{R}'$ for non-ground, tight logic programs. Users can use NEWGROUND by simply providing (tight) logic programs. Besides complete program translations, NEWGROUND supports partially reducability, such that rules can be explicitly excluded from the reduction. The general idea of the tool is to forward the output of NEWGROUND directly into user-chosen solver to compute any answer sets.

To test our prototype, we ran five benchmark sets comparing against state-of-the-art grounders GRINGO and IDLV. For comparison, we measure grounding sizes, grounding times and solving capabilities of the groundings. These preliminary experiments indicate that our body-decoupled grounding approach brings advantages throughout most of the benchmarks. However, we are aware of the strengths that sophisticated grounding systems bring, which is why we focused on partially reductions for rules with high body-density. So the results of our benchmarks do not imply the overall advantage of body-decoupled grounding, instead they suggest a well-working interaction between a reduction-approach and a traditional ground-and-solve system. This makes NEWGROUND a prime example for potentially incorporating into any traditional, state-of-the-art grounding system.

5.2 Related Work

ASP in its propositional case (ground) is crucial for the definition of answer-set semantics, however, it is the predicate (non-ground) version that simplifies modeling and makes the formalism an effective problem-solving technique. As defined in Chapter 2, non-ground programs require some form of grounding, the crucial task of instantiating the variables in non-ground rules, which is the reason for its harder complexity [BET11]. This is why grounding in general is an active research field and in literature there are several attempts to handle the problems that come with it [GLM$^+$18]. In the following, we present related works and attempts that have been made so far.

While there are many modern approaches, traditional ASP implementations typically follow the two-step evaluation process called ground-and-solve [KLPS16]. Syrjänen [Syr01] was one of the first releasing the grounder called LPARSE, a front-end grounder that accepts logic programs under the ω-restriction. A similar approach was used by the earlier versions of the well-known grounder GRINGO [GST07] using the λ-restriction, an extension to one the earlier mentioned. The restrictions are essentially used to handle positive recursion throughout predicates and ensure that the input programs have a finite grounding. However, more recent versions of GRINGO as well as the DLV grounding system [FLP12] follow a less restrictive approach called *safety*, the modern approach in this area, which requires that each variable in a rule occurs in some positive body literal. The grounding of these systems is typically an iterative bottom-up process guided by the expansion of the program's term base [GLM$^+$18].

[1] The system (incl. supplemental material) is available at `https://github.com/viktorbesin/newground`.

More recently, the DLV system was branched under the work of Calimeri et al. [CFPZ17] resulting in the grounder IDLV. While it is based on the same theoretical foundation as its predecessor, new optimizations and usability features, e.g. Python interface, annotations etc., were added.

Beyond these traditional grounding system literature have seen many approaches actually trying to mitigate the ASP grounding bottleneck. On the one hand, there are several ASP *extensions* that hybridize the ASP language with other formalisms, such as constraint programming [OS12, BL17] and difference logic [GKK+16, SL16]. These can be used to express any hard-to-ground constraints, which can then be efficiently evaluated using an appropriate solver interoperating with the ASP system, i.e. circumventing the grounding bottleneck by avoiding the grounding using a translation to another theory.

On the other hand, there are also several promising systems working on plain ASP, e.g. *lazy grounding* technique. The idea of lazy grounding is to instantiate non-ground rules on-demand when its body is satisfied, i.e. during solving. Thereby, the grounding of unnecessary rules, i.e. rules that are never satisfied, can be prevented which therefore mitigates the grounding bottleneck. To this end, lazy grounding has been implemented in different tools, e.g. GASP [PDPR09], ASPERIX [LN09] and ALPHA [Wei17]. Even though lazy grounding has performed well in preliminary results, later experiments have shown that their performance is still not competitive with state-of-the-art systems [GLM+18]. Further, later works by Cuteri et al. [CDRS17], compare two techniques *lazy instantiation* and *propagators* similar to lazy grounding. The input is simplified by omitting the problematic constraints, which are then either lazily instantiated if violated by a computed stable model or emulated using CDCL-based ASP solvers. However, this approach is based on procedures written in a imperative language, that is specific for the problem at hand, and therefore time-consuming. More recently, there were also approaches translating these non-ground constraints into dedicated C++ procedures instantiating the constraints automatically [CDRS19, CDRS20]. Preliminary results for these systems, which are built on the ASP solver WASP [ADLR15], show good performance compared to state-of-the-art ASP systems.

In addition, there are other approaches that restructure logic programs to circumvent the grounding bottleneck. While the body-decoupling of NEWGROUND is a result of the reduction used for grounding, the tool LPOPT [BMW16] focuses on optimizing non-ground programs directly before grounding. By splitting up problematic rules by the means of tree decomposition algorithms the combinatorial load can be significantly reduced as preliminary experiments have shown. This concept was later revisited by Calimeri et al. [CFPZ18] in combination with IDLV, however, experiments showed that the resulting groundings tend to be problematic for the tested solving modules WASP and CLASP [GKK+15].

To the best of our knowledge, this work and the tool NEWGROUND is the first deployment of the reduction between different types of logic programs for grounding. However, reductions have been used in the context of logic programs before. The tool SELP [BMW20] uses a

reduction from ground, epistemic logic programs to non-ground, disjunctive programs (under bounded arity) to solve ELP solely from ASP systems.

5.3 Future Work

This thesis introduces a new approach on how to handle the ASP grounding bottleneck and gives rise to plenty of future work.

While the preliminary results show that NEWGROUND can outperform state-of-the-art grounders in selected scenarios, there is still much room for testing our approach on more instances to further evaluate and improve it. However, as we already stated earlier, it is very unlikely that our approach beats highly optimized grounders in every imaginable scenario, which is why we think an integration of this technique into intelligent grounders makes sense. This way, heuristics can automatically estimate program parts that likely benefit from our body-decoupled grounding approach.

Besides the maintenance of NEWGROUND, the extension of the prototype to non-ground, normal programs seems very interesting. In fact, there is currently ongoing work focusing on the implementation and evaluation of this problem [Una22]. However, we believe that orderings might not be optimal as the encoding will lift the grounding size tremendously. Optimization and further evaluation are therefore still required in this area.

Driven by the development of truly competitive ELP solvers, we believe that there could lie potential in the reduction introduced for non-ground, disjunctive programs, especially, since the runtime evaluation shows even lower bounds than for the non-ground, tight translation and epistemic negations are only subjects of very simply rules. Therefore, the implementation and evaluation of a prototype for this approach might be interesting.

List of Figures

List of Tables

Listings

Bibliography

[ABC+15] Michael Abseher, Bernhard Bliem, Günther Charwat, Frederico Dusberger, and Stefan Woltran. Computing secure sets in graphs using answer set programming. *Journal of Logic and Computation*, 30(4):837–862, 08 2015.

[ADLR15] Mario Alviano, Carmine Dodaro, Nicola Leone, and Francesco Ricca. Advances in WASP. In Francesco Calimeri, Giovambattista Ianni, and Miroslaw Truszczynski, editors, *Logic Programming and Nonmonotonic Reasoning - 13th International Conference, LPNMR 2015, Lexington, KY, USA, September 27-30, 2015. Proceedings*, volume 9345 of *Lecture Notes in Computer Science*, pages 40–54. Springer, 2015.

[AS09] Gilles Audemard and Laurent Simon. Predicting learnt clauses quality in modern sat solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, IJCAI'09, page 399–404, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[BET11] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming at a glance. *Commun. ACM*, 54:92–103, 12 2011.

[BF91] Nicole Bidoít and Christine Froidevaux. Negation by default and unstratifiable logic programs. *Theoretical Computer Science*, 78(1):85–112, 1991.

[BHvMW09] Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009.

[BHW21] Viktor Besin, Markus Hecher, and Stefan Woltran. Utilizing Treewidth for Quantitative Reasoning on Epistemic Logic Programs. *Theory Pract. Log. Program.*, 21(5):575–592, 2021.

[BHW22a] Viktor Besin, Markus Hecher, and Stefan Woltran. Body-decoupled grounding via solving: A novel approach on the ASP bottleneck. In Luc De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022, Vienna, Austria, 23-29 July 2022*, pages 2546–2552. ijcai.org, 2022.

[BHW22b] Viktor Besin, Markus Hecher, and Stefan Woltran. Utilizing treewidth for quantitative reasoning on epistemic logic programs (extended abstract). In Lud De Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 5264–5268. International Joint Conferences on Artificial Intelligence Organization, 7 2022. Sister Conferences Best Papers.

[Bie08] Armin Biere. Picosat essentials. *J. Satisf. Boolean Model. Comput.*, 4(2-4):75–97, 2008.

[BL17] Marcello Balduccini and Yuliya Lierler. Constraint answer set solver EZCSP and why integration schemas matter. *Theory Pract. Log. Program.*, 17(4):462–515, 2017.

[BMW16] Manuel Bichler, Michael Morak, and Stefan Woltran. lpopt: A rule optimization tool for answer set programming. In Manuel V. Hermenegildo and Pedro López-García, editors, *Logic-Based Program Synthesis and Transformation - 26th International Symposium, LOPSTR 2016, Edinburgh, UK, September 6-8, 2016, Revised Selected Papers*, volume 10184 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2016.

[BMW20] Manuel Bichler, Michael Morak, and Stefan Woltran. selp: A Single-Shot Epistemic Logic Program Solver. *Theory Pract. Log. Program.*, 20(4):435–455, 2020.

[CDF+20] Francesco Calimeri, Carmine Dodaro, Davide Fuscà, Simona Perri, and Jessica Zangari. Efficiently coupling the I-DLV grounder with ASP solvers. *Theory Pract. Log. Program.*, 20(2):205–224, 2020.

[CDRS17] Bernardo Cuteri, Carmine Dodaro, Francesco Ricca, and Peter Schüller. Constraints, lazy constraints, or propagators in ASP solving: An empirical analysis. *Theory Pract. Log. Program.*, 17(5-6):780–799, 2017.

[CDRS19] Bernardo Cuteri, Carmine Dodaro, Francesco Ricca, and Peter Schüller. Partial compilation of ASP programs. *Theory Pract. Log. Program.*, 19(5-6):857–873, 2019.

[CDRS20] Bernardo Cuteri, Carmine Dodaro, Francesco Ricca, and Peter Schüller. Overcoming the grounding bottleneck due to constraints in ASP solving: Constraints become propagators. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1688–1694. ijcai.org, 2020.

[CFG+20] Pedro Cabalar, Jorge Fandinno, Javier Garea, Javier Romero, and Torsten Schaub. eclingo : A Solver for Epistemic Logic Programs. *Theory Pract. Log. Program.*, 20(6):834–847, 2020.

[CFPZ17] Francesco Calimeri, Davide Fuscà, Simona Perri, and Jessica Zangari. I-DLV: the new intelligent grounder of DLV. *Intelligenza Artificiale*, 11(1):5–20, 2017.

[CFPZ18] Francesco Calimeri, Davide Fuscà, Simona Perri, and Jessica Zangari. Optimizing answer set computation via heuristic-based decomposition. In Francesco Calimeri, Kevin W. Hamlen, and Nicola Leone, editors, *Practical Aspects of Declarative Languages - 20th International Symposium, PADL 2018, Los Angeles, CA, USA, January 8-9, 2018, Proceedings*, volume 10702 of *Lecture Notes in Computer Science*, pages 135–151. Springer, 2018.

[CPZ19] Francesco Calimeri, Simona Perri, and Jessica Zangari. Optimizing Answer Set Computation via Heuristic-Based Decomposition. *Theory Pract. Log. Program.*, 19(4):603–628, 2019.

[CRR19] Bernardo Cuteri, Kristian Reale, and Francesco Ricca. A logic-based question answering system for cultural heritage. In Francesco Calimeri, Nicola Leone, and Marco Manna, editors, *Logics in Artificial Intelligence - 16th European Conference, JELIA 2019, Rende, Italy, May 7-11, 2019, Proceedings*, volume 11468 of *Lecture Notes in Computer Science*, pages 526–541. Springer, 2019.

[DEGV01] Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.

[EFFW07] Thomas Eiter, Wolfgang Faber, Michael Fink, and Stefan Woltran. Complexity results for answer set programming with bounded predicate arities and implications. *Annals of Mathematics and Artificial Intelligence*, 51(2-4):123–165, 2007.

[EG95] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3–4):289–323, 1995.

[EGM97] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Trans. Database Syst.*, 22(3):364–418, 1997.

[EP06] Thomas Eiter and Axel Polleres. Towards automated integration of guess and check programs in answer set programming: a meta-interpreter and applications. *Theory Pract. Log. Program.*, 6(1-2):23–60, 2006.

[Fag94] François Fages. Consistency of Clark's completion and existence of stable models. *Logical Methods in Computer Science*, 1(1):51–60, 1994.

[FFS+18] Andreas A. Falkner, Gerhard Friedrich, Konstantin Schekotihin, Richard Taupe, and Erich Christian Teppan. Industrial Applications of Answer Set Programming. *Künstliche Intell.*, 32(2-3):165–176, 2018.

[FLP12] Wolfgang Faber, Nicola Leone, and Simona Perri. The intelligent grounder of DLV. In Esra Erdem, Joohyung Lee, Yuliya Lierler, and David Pearce, editors, *Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz*, volume 7265 of *Lecture Notes in Computer Science*, pages 247–264. Springer, 2012.

[Gel91] Michael Gelfond. Strong introspection. In Thomas L. Dean and Kathleen R. McKeown, editors, *Proceedings of the 9th National Conference on Artificial Intelligence, Anaheim, CA, USA, July 14-19, 1991, Volume 1*, pages 386–391. AAAI Press / The MIT Press, 1991.

[GKK+11] Martin Gebser, Benjamin Kaufmann, Roland Kaminski, Max Ostrowski, Torsten Schaub, and Marius Schneider. Potassco: The potsdam answer set solving collection. *AI Commun.*, 24(2):107–124, apr 2011.

[GKK+15] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Javier Romero, and Torsten Schaub. Progress in clasp series 3. In Francesco Calimeri, Giovambattista Ianni, and Miroslaw Truszczynski, editors, *Logic Programming and Nonmonotonic Reasoning*, pages 368–383, Cham, 2015. Springer International Publishing.

[GKK+16] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory solving made easy with clingo 5. In Manuel Carro, Andy King, Neda Saeedloei, and Marina De Vos, editors, *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA*, volume 52 of *OASIcs*, pages 2:1–2:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[GKKS19] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot ASP solving with clingo. *Theory Pract. Log. Program.*, 19(1):27–82, 2019.

[GKS11] Martin Gebser, Roland Kaminski, and Torsten Schaub. Complex optimization in answer set programming. *Theory Pract. Log. Program.*, 11(4-5):821–839, 2011.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9:365–385, 1991.

[GLM+18] Martin Gebser, Nicola Leone, Marco Maratea, Simona Perri, Francesco Ricca, and Torsten Schaub. Evaluation techniques and systems for answer set programming: a survey. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 5450–5456. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[GSS21] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability - Second Edition*, volume 336 of *Frontiers in Artificial Intelligence and Applications*, pages 993–1014. IOS Press, 2021.

[GST07] Martin Gebser, Torsten Schaub, and Sven Thiele. Gringo : A new grounder for answer set programming. In Chitta Baral, Gerhard Brewka, and John S. Schlipf, editors, *Logic Programming and Nonmonotonic Reasoning, 9th International Conference, LPNMR 2007, Tempe, AZ, USA, May 15-17, 2007, Proceedings*, volume 4483 of *Lecture Notes in Computer Science*, pages 266–271. Springer, 2007.

[Jan06] Tomi Janhunen. Some (in)translatability results for normal logic programs and propositional theories. *Journal of Applied Non-Classical Logics*, 16(1-2):35–86, 2006.

[JN16] Tomi Janhunen and Ilkka Nimelä. The answer set programming paradigm. *AI Magazine*, 37(3):13–24, Oct. 2016.

[KL99] Hans Kleine Büning and Theodor Lettmann. *Propositional logic - deduction and algorithms*, volume 48 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 1999.

[KLPS16] Benjamin Kaufmann, Nicola Leone, Simona Perri, and Torsten Schaub. Grounding and solving in answer set programming. *AI Mag.*, 37(3):25–32, 2016.

[KWB+15] Patrick Thor Kahl, Richard Watson, Evgenii Balai, Michael Gelfond, and Yuanlin Zhang. The language of epistemic specifications (refined) including a prototype solver. *J. Log. Comput.*, 25, 2015.

[LN09] Claire Lefèvre and Pascal Nicolas. The first version of a new ASP solver : Asperix. In Esra Erdem, Fangzhen Lin, and Torsten Schaub, editors, *Logic Programming and Nonmonotonic Reasoning, 10th International Conference, LPNMR 2009, Potsdam, Germany, September 14-18, 2009. Proceedings*, volume 5753 of *Lecture Notes in Computer Science*, pages 522–527. Springer, 2009.

[LZ03] Fangzhen Lin and Jicheng Zhao. On tight logic programs and yet another translation from normal logic programs to propositional logic. In Georg Gottlob and Toby Walsh, editors, *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*, pages 853–858. Morgan Kaufmann, 2003.

[Mor19] Michael Morak. Epistemic logic programs: A different world view. In Bart Bogaerts, Esra Erdem, Paul Fodor, Andrea Formisano, Giovambattista Ianni, Daniela Inclezan, Germán Vidal, Alicia Villanueva, Marina De Vos,

and Fangkai Yang, editors, *Proceedings 35th International Conference on Logic Programming (Technical Communications), ICLP 2019 Technical Communications, Las Cruces, NM, USA, September 20-25, 2019*, volume 306 of *EPTCS*, pages 52–64, 2019.

[MT91] Wiktor Marek and Mirosław Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38(3):588–619, 1991.

[OS12] Max Ostrowski and Torsten Schaub. ASP modulo CSP: The clingcon system. *Theory Pract. Log. Program.*, 12(4-5):485–503, 2012.

[Pap94] Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[PDPR09] Alessandro Dal Palù, Agostino Dovier, Enrico Pontelli, and Gianfranco Rossi. GASP: answer set programming with lazy grounding. *Fundam. Informaticae*, 96(3):297–322, 2009.

[Pos44] Emil L. Post. Recursively enumerable sets of positive integers and their decision problems. *Bulletin of the American Mathematical Society*, 50(5):284 – 316, 1944.

[Rog87] Hartley Rogers. *Theory of Recursive Functions and Effective Computability.* MIT Press, Cambridge, MA, USA, 1987.

[SE16] Yi-Dong Shen and Thomas Eiter. Evaluating epistemic negation in answer set programming. *Artif. Intell.*, 237:115–135, 2016.

[SL16] Benjamin Susman and Yuliya Lierler. Smt-based constraint answer set solver EZSMT (system description). In Manuel Carro, Andy King, Neda Saeedloei, and Marina De Vos, editors, *Technical Communications of the 32nd International Conference on Logic Programming, ICLP 2016 TCs, October 16-21, 2016, New York City, USA*, volume 52 of *OASIcs*, pages 1:1–1:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In Alfred V. Aho, Allan Borodin, Robert L. Constable, Robert W. Floyd, Michael A. Harrison, Richard M. Karp, and H. Raymond Strong, editors, *Proceedings of the 5th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1973, Austin, Texas, USA*, pages 1–9. ACM, 1973.

[SNS02] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artif. Intell.*, 138(1-2):181–234, 2002.

[SW18] Torsten Schaub and Stefan Woltran. Special issue on answer set programming. *Künstliche Intell.*, 32(2-3):101–103, 2018.

[Syr01] Tommi Syrjänen. Omega-restricted logic programs. In Thomas Eiter, Wolfgang Faber, and Miroslaw Truszczynski, editors, *Logic Programming and Nonmonotonic Reasoning, 6th International Conference, LPNMR 2001, Vienna, Austria, September 17-19, 2001, Proceedings*, volume 2173 of *Lecture Notes in Computer Science*, pages 267–279. Springer, 2001.

[Tru11] Miroslaw Truszczynski. Revisiting epistemic specifications. In *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning*, volume 6565 of *LNCS*, pages 315–333. Springer, 2011.

[Tur39] Alan M. Turing. Systems of logic based on ordinals. *Proceedings of the London Mathematical Society*, Series 2 - Vol.45:161–228, 1939.

[Una22] Kaan Unalan. Body-Decoupled Grounding in Normal Answer Set Programs. Bachelor's Thesis, Faculty of Informatics, TU Wien, Austria, 2022.

[Val79] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[Wei17] Antonius Weinzierl. Blending lazy-grounding and CDNL search for answer-set solving. In Marcello Balduccini and Tomi Janhunen, editors, *Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings*, volume 10377 of *Lecture Notes in Computer Science*, pages 191–204. Springer, 2017.

[Wra76] Celia Wrathall. Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

[WTF20] Antonius Weinzierl, Richard Taupe, and Gerhard Friedrich. Advancing Lazy-Grounding ASP Solving Techniques - Restarts, Phase Saving, Heuristics, and More. *Theory Pract. Log. Program.*, 20(5):609–624, 2020.