FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Realising argumentation using answer set programming and quantified boolean formulas

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

MSc Martin Diller
Registration Number 01228429

to the Faculty of Informatics

at the TU Wien

Advisor: Ao. Univ-Prof. Dr. Uwe Egly
Second advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Woltran

The dissertation has been reviewed by:

Federico Cerutti

Matthias Thimm

Vienna, 18th February, 2019

Martin Diller

Erklärung zur Verfassung der Arbeit

MSc Martin Diller
Miesbachgasse 7, 1020 Wien

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, 18. Februar 2019

Martin Diller

Acknowledgements

First of all, thank you to my advisors Uwe Egly and Stefan Woltran for their support and great disposition as well as humour throughout work on my PhD. Special thanks to Stefan for sharing his always pertinent insights with me in key moments, giving me a place in the ADF/GRAPPA team, and connecting me to the argumentation community at large.

I am indebted to the LogiCS program (and Austrian Science Fund) for giving me the opportunity to do my PhD at TU Wien; in particular, to its founder Helmut Veith who is greatly missed and Anna Prianichnikova for her full-hearted engagement. Also to the Institute of Information Systems team at TU Wien for their extremely friendly and always helpful support. Thank you Juliane Auerböeck, Beatrix Forsthuber, Eva Nedoma, and Toni Pisjak!

I am grateful for the collaborations and quality time spent with everyone from the ADF/GRAPPA team and extended family. Particularly heartfelt thank you goes to Adrian Haret, Atefeh Keshavarzi-Zafarghandi, Thomas Linsbichler, Sylwia Polberg, and Johannes P. Wallner. Thanks also Anna Rapberger for helping out at the last moment. A big thank you goes to Florian Lonsing for his assistance with all matters QSAT and beyond.

I am thankful to Anthony Hunter for being a wonderful host in London during winter of 2016. I have especially fond memories of our afternoon chats, from which I also learned a lot. A very warm thank you goes to Adam Z. Wyner (and Anna Havinga) for making me feel more than welcome in Aberdeen during winter of 2017. Again to Adam and also Hannes Strass for trusting me to join them in working at the core of the EMIL project. Related to the latter, I am likewise in debt with the Attempto team from the University of Zürich for making their valuable knowledge and tools available online.

I am very grateful to time spent with old and new-found friends in Vienna and elsewhere during the last 4 years. Shqiponja Ahmetaj and Labinot Bajraktari, who more than office-mates have been like substitute siblings during this time, I cannot leave without mentioning explicitly here.

Thank you beautiful Elena for bringing so much vitality and enchantment into my life this last year.

A very special thank you goes to my mother, Ana, and father, Guillermo, for their unconditional and very loving support these last years as before. Also, to my exceptional sister Carolina who I can always count on to be by my side.

Finally, I want to commemorate my dear grandmother, Marcela, who has left for a better place when I was commencing my PhD. She has left many greatly missing her yet, as she certainly desired, also joyful and admiring when reminded of her extraordinary good nature and free spirit.

Kurzfassung

Einer der wichtigsten Schlüsselfaktoren menschlicher Kommunikation ist die Argumentation, womit individuelle oder kollektive Entscheidungen zu komplexen Themen auch in Anbetracht unvollständiger oder widersprüchlicher Informationen getroffen werden können. Artifizielles und rechnergestütztes Argumentieren wird daher zu einer immer bedeutungsvolleren Teildisziplin der Künstlichen Intelligenz, mit dem Ziel, menschliche Argumentation zu unterstützen oder sogar zu automatisieren und Computersysteme so zu verbessern, dass sie Argumente generieren und auswerten können.

Diese Arbeit beschäftigt sich mit der Implementierung allgemeiner formaler Modelle grundlegender Argumentationssysteme. Ihnen gemeinsam ist die hohe Berechnungskomplexität - im Falle der Argumentationsprobleme, die wir untersuchen, können diese von vollständig für die ersten Stufen der Polynomialzeithierarchie bis zu grundsätzlich unentscheidbar reichen. Ein Standardverfahren zur Lösung solcher komplexen Rechenprobleme ist es, Translationsmöglichkeiten in andere Formalismen, bei denen effiziente Systeme existieren, zur Verfügung zu stellen. Aufgrund der Komplexität unserer Aufgabe betrachten wir quantifizierte boolsche Formeln (QBF) und “Answer Set Programming” (ASP) als Zielformalismen. Ersteres ist eine Verallgemeinerung von Aussagenlogik, welche Unterklassen von Formeln hat, deren Komplexität vollständig für jede Stufe der Polynomialzeithierarchie ist. “Answer Set Programming” ist ein Paradigma für deklarative Problemlösung, die aus der Logikprogrammierung stammt und allgemein unentscheidbar ist, jedoch wichtige entscheidbare Fragmente besitzt.

Konkret entwickeln wir Methoden zur Implementierung zentraler Argumentationsprobleme für “abstract dialectical frameworks” und “defeasible theories”. “Abstract dialectical frameworks” (ADFs) ist einer der umfassendsten graphischen Formalismen zur Evaluierung von Argumentationsszenarien, wobei die Knoten Aussagen oder Behauptungen repräsentieren und mit Akzeptanzkonditionen assoziiert sind, die komplexe Beziehungen zwischen den Aussagen wie zum Beispiel Ablehnung oder Unterstützung repräsentieren. Auf Basis solcher Repräsentationen liefern verschiedene Semantiken Konfliktlösungmechanismen. Andererseits erlauben “defeasible theories” Sammlungen von strikten und annullierbaren Regeln erster Ordnung auszudrücken. Potentielle Widersprüche zwischen den annullierbaren Regeln werden durch die “direct stable semantics” entschärft, wodurch auch die Möglichkeit entsteht, Argumente für spezifische Aussagen auf Basis der durch eine “defeasible theory” ausgedrückten Informationen zu generieren.

Zu Beginn entwickeln wir in dieser Arbeit komplexitätsberücksichtigende QBF Kodierungen für “stable semantics” für ADFs als auch verknüpfungs-informations-sensible QBF-Kodierungen für alle der wichtigsten Semantiken für ADFs. Letztere profitieren von der Information der Verknüpfungsarten (Beziehung zwischen den Netzknoten) der ADFs wenn vorhanden; der Grund dafür ist, dass die QBF Solver in der Lage sind, diese Information im gleichen Ausmaß zu nützen wie die Verfügbarkeit der Information die Komplexität der Argumentationsaufgaben, die wir studieren, vereinfacht. Des Weiteren entwickeln wir neue dynamische ASP Kodierungen für die wichtigsten Semantiken für ADFs. Wir machen davon Gebrauch, dass die kombinierte Komplexität vom Schließen in ASP Programmen mit Prädikaten mit beschränkter Stelligkeit genau mit der Komplexität der Argumentationsaufgaben, die wir untersuchen, übereinstimmt.

Schließlich untersuchen wir die Implementierung des gesamten Prozesses von der Auswertung von Sammlungen strikter und annullierbarer Regeln, ausgedrückt in einer prominenten “Controlled Natural Language” (Kontrollierte Natürliche Sprache) (CNL), ACE, bis hin zu der Translation solcher Regeln in “defeasible theories” und deren Evaluation durch “direct stable semantics”. Dies beinhaltet das Studieren der Translation der in CNL ausgedrückten Regeln zu “defeasible theories”; die Kodierung allgemeiner Regeln, die existenzielle Quantifizierung beinhalten, in normale (ohne existenzielle Quantifizierung) Regeln; als auch die Evaluation von “defeasible theories” durch ASP Kodierungen. Wir haben Prototypsysteme für alle von uns entwickelten Strategien produziert und berichten auch über vorläufige empirische Evaluationen.

Abstract

Argumentation is one of the key manners in which humans individually and collectively make sense of complex scenarios about which the information that is available is incomplete or even inconsistent. Computational or artificial argumentation is thus also an increasingly important sub-field of AI aiming at supporting or even automating human argumentation as well as enhancing computational systems with means of generating and evaluating arguments.

This work is concerned with the implementation of general formal models underlying computational argumentation systems. What most of such models have in common is their high computational complexity; in the case of the reasoning problems we study, these range from being complete for the first levels of the polynomial hierarchy to being undecidable in general. A standard way of addressing computational problems with such high complexity is by providing translations to other formalisms for which efficient systems exist. Given the complexity of the tasks we address, in this work we consider quantified boolean formulas (QBFs) and answer set programming (ASP) as target formalisms. The first being a generalisation of propositional logic having sub-classes of formulas whose complexity is complete for each level of the polynomial hierarchy. Answer set programming, on the other hand, is a paradigm for declarative problem solving having sprung out of logic programming and being undecidable in general, yet having important decidable fragments.

Concretely, we develop means of implementing some of the main reasoning problems for abstract dialectical frameworks and defeasible theories. Abstract dialectical frameworks (ADFs) are one of the most comprehensive graphical formalisms for evaluating argumentation scenarios, with nodes representing statements or assertions and acceptance conditions associated to the nodes specifying complex relations, such as attack and support, between the statements. Different semantics provide conflict resolution mechanisms on the basis of such an abstract representation. Defeasible theories, on the other hand, allow for expressing collections of strict and defeasible first order rules. Potential conflicts between the defeasible rules are resolved via the direct stable semantics, which also provides means of generating arguments for particular claims on the basis of the information expressed in a defeasible theory.

In this work we, first of all, develop complexity sensitive QBF encodings for the stable semantics for ADFs as well as link information sensitive QBF encodings for all the major

semantics for ADFs. The latter make use of information about the link (relationships between the nodes) types of ADFs whenever available; the motivation being that solvers for QBFs are able to make use of this information to the same degree as which availability of the information can make the complexity of the reasoning tasks we study easier. We secondly develop novel dynamic ASP encodings for the main semantics for ADFs. We make use of the fact that the combined complexity of reasoning on ASP programs with predicates of bounded arity size exactly matches the complexity of the reasoning tasks we investigate for ADFs.

Finally, we study the implementation of the whole pipeline of evaluating collections of strict and defeasible rules expressed in a prominent controlled natural language (CNL), ACE, to the translation of such rules into defeasible theories and their evaluation via the direct stable semantics. This involves studying the translation of the rules expressed in the CNL into defeasible theories, the encoding of general rules involving existential quantification into normal (i.e. not having existential quantification) rules, as well as the evaluation of defeasible theories via the direct stable semantics using ASP encodings. We have produced prototype systems for all of the implementation strategies we devise in this work and also report on preliminary empirical evaluations.

Contents

CHAPTER 1

Introduction

Argumentation and its implementation in AI systems Broadly construed the study of argumentation is concerned with how assertions are (or should be) "proposed, discussed, and resolved in the context of issues upon which several diverging opinions may be held" [BD07]. It is today a multidisciplinary field involving especially philosophy, law, and AI.

This work is more specifically concerned with the implementation of models of argumentation as developed in the subfield of AI dubbed "computational models of argumentation" or more recently and somewhat paradoxically "formal argumentation". Among the disciplines concerned with argumentation this is, perhaps, the most exciting due to its reliance on precise concepts and theories on the one hand (often motivated by developments of neighboring disciplines or applications) and their realisation in computational systems on the other. The latter not only allow getting a handle on the computational and material adequacy of the formal models, but also have the potential of directly influencing the quality and expanding the reach of argumentation in the context of the increasingly ubiquitous cyberspace. Promising application areas are e.g. in health-care, e-governance, law, and the web (see [ABG$^+$17] for an overview).

Formal argumentation takes up some of the preoccupations of the informal logic movement initiated in the 1950's in philosophy [Gro17]. Specifically, it acknowledges the limitations of using the tools of modern formal logic as initiated with the work of Frege in 1879 [vH02], with its focus on the notion of mathematical proof, to model everyday reasoning. Indeed, classical deductive logic is pushed to its limits when used for capturing reasoning on the basis of incomplete, uncertain, and conflicting information; a type of reasoning which must be revisable. On the other hand, rather than abandoning formal tools altogether, formal argumentation is an attempt at developing more adequate formal tools and theories for the purpose of apprehending reasoning modes beyond those used in constructing mathematical proofs[1]. Also, computational concepts, techniques, and concerns play an increasingly central role. See [vEV18] for a recent survey of the history of formal argumentation in the broader context of argumentation theory and [Pra18] for a survey more focused on the developments in AI.

Argumentation can be viewed statically in that all information relevant to argumentation is fixed. For instance in a medical setting as in [HW12], when constructing arguments for recommending one treatment over another to a specific patient, all the relevant evidence regarding the effects of treatments and the information about the patient can be fixed to that available at a specific time. In the same manner, when studying the arguments contained in product reviews posted on a web-site of interest (see, for instance, [WSAB12]), it is reasonable to focus on the arguments available at a certain date. In static scenarios it is also typical to adopt the "third person view" of the analyst studying a discussion and assume the analyst has access to all the relevant information available at that time; i.e. the information is "global".

The static perspective is the more usual analytical perspective in formal argumentation, which usually focuses on argumentation as a form of "inference" [Pra18]. Much of its impetus comes from developments, especially during the 80's, in logic programming and non-monotonic reasoning [BNT08]. We also take up this perspective in our work and therefore henceforth, except if stated otherwise, when we refer to formal argumentation we have this restricted viewpoint in mind. An alternative yet often complementary[2] viewpoint is that taken up in dialogical models of argumentation (see e.g. [MP09] and again [Pra18] for a historical overview), where information is dynamic and distributed over agents.

Even within the study of argumentation as a form of inference there are meanwhile quite diverse theoretical frameworks, with the main division being between structured and abstract approaches to argumentation. Structured argumentation formalisms, as the name suggests, offer means of modelling the structure of arguments; e.g. premisses, inference rules, and claims. Usually this is done by assuming an underlying logic; for instance, classical logic [BH01] or default logic [BDKT97]. Prominent examples of structured argumentation formalisms are assumption based argumentation [CFST18], deductive argumentation [BH18], defeasible logic programming [GS18], and ASPIC$^+$ [MP18].

In abstract argumentation the notion of argument is usually taken as a primitive and the focus is on the relationships, typically that of attack, between the arguments. The insight of the landmark work of [Dun95], which initiated this line of research and also marks the beginning of rise to prominence of formal argumentation within AI, is that the acceptability of arguments (whether an argument is defeated or not) in the context of a discussion can be decided solely on the basis of the relationships between the arguments in the debate. More concretely, Dung defined several semantics formalising diverse criteria to

[1]And even in these, see e.g. [PLB$^+$17].

[2]The state of a dialogue can be fixed at a certain time and hence the static perspective comes into play.

determine the acceptability of arguments based on their relationship with other arguments in a discussion. Arguments that can be accepted together (w.r.t. a semantics) form so called extensions. To date a plethora of semantics exist even for Dung's argumentation frameworks (see [BCG18]) and much research in formal argumentation is devoted to studying properties [Bau18] and relationships between the semantics [vdTV18].

The exact relation between structured and abstract argumentation is somewhat of a controversial issue. Since the work of Dung, who himself used abstract argumentation to reconstruct and compare several then important non-monotonic logics and logic programming formalisms, many researchers in structured argumentation have themselves used abstract argumentation to define the semantics of their formalisms. This leads to what is often referred to as the instantiation pipeline for evaluating structured argumentation [CA07]. The pipeline in question starts of with a knowledge base with formulas in some base logic. Arguments are constructed from the knowledge base and then relationships (usually that of attack; e.g. one argument attacks another if its conclusion is the negation of one of the premisses of the latter) between the arguments are defined. This leads to an abstract representation of the arguments in the form of a graph: in the context of Dung's argumentation frameworks (AFs) the nodes are the arguments and the directed edges the attacks. These argumentation graphs can be evaluated using abstract argumentation semantics. An important issue in the study of the instantiation of structured argumentation using abstract argumentation is that the instantiation satisfies some basic criteria codified into so called rationality postulates [Cam18], e.g. that the extensions obtained when evaluating the instantiated argumentation graph are consistent (in terms of the base logic).

On the other hand, there are structured argumentation formalisms (most notably, defeasible logic programming as presented in e.g. [GS04]) which do not conform to any of Dung's semantics. Also, there are structured argumentation formalisms like assumption-based-argumentation [CFST18] which can be evaluated in a direct approach that is equivalent to the instantiation approach but does not require the construction of argument graphs. More fundamentally, in the area of abstract argumentation there has been a surge of proposals of extensions to Dung's initial argument frameworks (composed of arguments and attacks) which more often than not attempt to directly model natural language arguments (rather than first resorting to structured argumentation) using graphical formalisms. Some examples of elements that have been added to Dung's argumentation frameworks include preferences, values, the support relation, set-attacks, attacks on attacks, weights on attacks, and constraints. See [BPW14] for a survey of extensions of abstract argumentation formalisms, [PW18] for a recent critique of this line of work, and [Mod13] for a plausible line of defense. Many systems for argumentation also use abstract argumentation as a knowledge representation formalism; particularly notable is that also research in argument mining [CV18] (mining of arguments from texts, e.g. on social-media or elsewhere on the web), which is one of the current major driving forces behind research in argumentation, often assumes some form of graphical representation as its output.

What most formalisms in argumentation have in common is their high computational complexity; see [DD18] for an overview. Indeed, structured argumentation includes formalisms which can capture classical logic [BH01] and its consequence relation is, hence, undecidable in general [Tur37]. But even the complexity of some of the basic decision problems, e.g. deciding whether an argument is contained in one or all extensions (w.r.t. a semantics), for abstract argumentation in Dung's rather simple framework spans the first two levels of the polynomial hierarchy [DD18]. It should thus come as no surprise that much research has gone into computational techniques and systems for evaluating argumentation formalisms. Dung's argumentation frameworks (AFs) in particular have received the most attention (see [CDG$^+$15] for an early overview) as witnessed by the fact that the second international competition on computational models of arguments (ICCMA'17)[3] featured 16 solvers for Dung's AFs, with at least 9 solvers participating in each of the 24 tasks resulting from the combination of considered semantics and reasoning problems.

Implementation techniques can be broadly classified in direct and reduction based, where the direct approach involves the development of native algorithms for the formalism and reasoning problem of interest. The reduction approach is based on the translation of the reasoning problem of interest to some formalism for which systems exist; most notably SAT and QSAT [BHvMW09], constraint satisfaction problems [RvBW06], and answer-set programming [SW18b]. For structured argumentation another difference is whether the implementation makes use of the instantiation to abstract argumentation (which allows making use of systems for abstract argumentation) or not. See [CGTW18] for a recent survey on implementations of formal argumentation formalisms, including also a review of some of the more important general purpose systems (e.g. for annotating texts with argument components or providing support for critical thinking and debate) available for argumentation.

Frameworks for argumentation and target formalisms for implementations considered in this work In this study we investigate the issue of implementing reasoning for two rather novel yet also quite general frameworks for abstract and structured argumentation respectively: abstract dialectical frameworks and the direct-stable-semantics for defeasible theories. Moreover, in the latter case we consider the entire pipeline from a knowledge base expressed in a controlled natural language to the evaluation of the knowledge base using the argumentation-based semantics. We present reduction based implementation techniques based on quantified boolean formulas and answer-set-programming.

Abstract dialectical frameworks (first defined in [BW10] and later refined in [BES$^+$13]) or ADFs for short are one of the most comprehensive extensions of Dung's abstract argumentation frameworks, allowing complex relationships between nodes to be specified by associating acceptance conditions in the form of propositional formulas to the nodes. Depending on the level of abstraction, the nodes can represent arguments, statements, or

[3]See http://argumentationcompetition.org/index.html.

even elements from other domains for which conflict-resolution using argumentation-based semantics makes sense. ADFs allow representing several of the most important existing extensions of Dung's argumentation frameworks [Pol16, Pol17] and have originally been conceived as a form of "argumentation middle-ware" [BES$^+$13] but have also been used directly for modelling e.g. in applications in argument-mining [CV16], law [AAB16], and general purpose debating technology [Neu18]. The most important semantics for Dung's frameworks have been generalised to ADFs [BES$^+$13], with the complexity of the associated decision problems usually being one level higher in the polynomial hierarchy than the same problems for Dung's AFs [SW15]. I.e. complexity for ADFs spans the first three levels of the polynomial hierarchy. We refer to [BES$^+$18] for a thorough discussion of the motivations and current theory built around ADFs.

The direct stable semantics for defeasible theories is the outcome of previous investigations [WBDC15, Str18] on defining a semantics for knowledge bases consisting of propositional strict and defeasible rules via abstract argumentation. The authors in [SW17] rather opt for an approach, where arguments for particular claims are an optional by-product rather than part of the semantics. The motivations behind the direct stable semantics are thus, in the first place, to provide means of evaluating knowledge bases of strict and defeasible rules inspired in developments in formal argumentation. Defeasible rules are usually thought of as "normality assumptions" (i.e. as in the natural language expressions "normally', "it is typical that" or "usually"); moreover, there may be conflicts between the defeasible rules. Crucially, the direct stable semantics satisfies some of the main rationality postulates defined in the literature [CA07] by construction, while also avoids several computational problems of evaluating knowledge bases via the instantiation process. Finally, the semantics is also defined for defeasible theories having some first order elements and thus more suitable for capturing arguments expressed using natural language [WS17]. Already for propositional theories acceptance problems (i.e. deciding whether an atom is contained in some/all stable set/s) are complete for the second level of the polynomial hierarchy, while the complexity of first order defeasible reasoning, although yet not made precise, can be expected to be much harder.

Turning to the formalisms which we use as target for the realisations we devise in this work, quantified boolean formulas (QBFs) [KB09] are an extension of propositional logic with subclasses of QBFs identifiable by their syntactic structure (their prefix when in the prenex-normal-form) capturing each level of the polynomial hierarchy. More precisely, for each level of the polynomial hierarchy there is a subclass of QBFs whose satisfiability problem (QSAT) is complete for that level. Answer-set-programming (ASP) [SW18b] on the other hand is an increasingly influential paradigm for declarative problem solving having roots in logic programming. The main decision problems for propositional ASP span the first two levels of the polynomial hierarchy, for first order ASP they are up to NEXPTIME$^{\text{NP}}$-complete, while allowing functional symbols in ASP programs gives ASP the expressive power of classical logic. Beyond the facts that the expressive power of QBFs and ASP is at the right level for capturing the decision problems we study in this work, these formalisms are of interest to us because of the impressive reasoners that exist for them. See, for instance, the webpages of the recent international competitions for QBF[4] and ASP[5] solvers.

Contributions and structure of the thesis Concretely, in this work we first develop novel QBF and ASP encodings for the main reasoning problems for ADFs. We start of with QBF encodings in Section 3.1. First, in Section 3.1.2 we build on previous work of ours [Dil14] to give complexity-sensitive QBF encodings for the main reasoning problems for ADFs w.r.t. the stable semantics. Note that in [Dil14] we did not consider this more complex semantics. Also, ours are the only QBF encodings for ADFs we are aware of to date. We thus continue the tradition of developing QBF encodings for argumentation problems initiated for AFs in [EW06] and [AC13] (although these have not produced systems).

Links of ADFs (edges between nodes) can be either supporting, attacking, redundant or dependent; with the complexity of reasoning for ADFs with a number of links that are neither attacking nor supporting bounded by a fixed constant roughly dropping one level of the polynomial hierarchy with respect to general ADFs [SW15]; in fact, this result can be easily generalised to links that are neither attacking nor supporting or, alternatively, whose type is unknown (determining the link-type is coNP-complete) as we sketch in Section 3.1.3.1. We present link information sensitive QBF encodings for all of the main semantics (admissible, complete, preferred, grounded, stable) and reasoning problems (verification, as well as credulous and skeptical acceptance problems) for ADFs in Section 3.1.3. The encodings we develop make use of additional information about the link types of the ADFs, whenever this information is available. The motivation behind the latter is that the additional information about the links may serve solvers for QBFs being fed an implementation of our encodings to the same degree that the extra information may make the reasoning tasks easier.

We present a re-implementation (with some improvements in the design for purposes of inspection) and extension of our previous QBF-based system for ADFs from [Dil14, DWW14], `QADF`[6], in Section 3.1.4. The system now also produces link-information-sensitive encodings for the admissible, preferred, and stable semantics. We also discuss results of a preliminary investigation on the effect of using different QBF-solvers and preprocessors on our link-information-sensitive encodings for the admissible and preferred semantics.

While we are responsible for the only QBF-based system for ADFs we are aware of, the first system for ADFs was the ASP-based system `ADFSys` [EW12] which lead to the (likewise ASP-based) `DIAMOND` (DIAlectical FraMewOrks eNcoDings) family of systems [ES14, ES16, SE17] (for AFs there are several ASP-based systems; see [DDP$^+$18]). These systems have in common that they rely on static encodings, i.e. the encoding does not change for different framework instances; this approach is hence limited by

[4]http://www.qbflib.org/index_eval.php
[5]http://aspcomp2017.dibris.unige.it/
[6]https://www.dbai.tuwien.ac.at/proj/adf/qadf/

the data complexity of ASP (which only reaches the second level of the polynomial hierarchy [EG95, EGM97]). Therefore, preferred semantics in particular (which comprise the hardest problems for ADFs) need a more complicated treatment involving two consecutive calls to ASP solvers with a possibly exponential blowup for the input of the second call.

In Section 3.2, we introduce a new method for implementing reasoning tasks for ADFs (acceptance problems for the admissible, complete, preferred, grounded, and stable semantics) such that even the hardest among the problems are treated with a single call to an ASP solver. Our approach makes use of the fact that the combined complexity of ASP for programs with predicates of bounded arity [EFFW07] exactly matches the complexity of ADFs. This approach is called dynamic, because the encodings are generated individually for every instance. This allows to generate rules of arbitrary length that can take care of NP-hard subtasks themselves. This particular method has been advocated in [BMW16b] in combination with tools that decompose such long rules whenever possible in order to avoid huge groundings [BMW16a]. To the best of our knowledge, our work is the first to apply this technique in the field of argumentation.

We discuss how to make use of our dynamic technique in the context of the encodings developed for the `DIAMOND` systems in Section 3.2.6. In Section 3.2.7 we present an implementation of the encodings (for the admissible, complete, preferred, and stable semantics) in our system `YADF`[7] and discuss experiments we carried out to compare the performance of our implementation to some of the other main existing systems for ADFs.

A defining characteristic of argumentation is that it is expressed using natural language. Much of current research in argument-mining focuses on extracting components (e.g. claims, premisses) and relations between arguments (e.g. attack, support) using shallow natural language processing techniques and disregarding reasoning on the basis of the extracted elements. On the other hand, models of formal argumentation more often than not ignore the issue of their applicability to information expressed in natural language. Something of a middle ground is offered by tying a formal model for argumentation to a controlled natural language (CNL) [Kuh14][8]: a restricted subset of natural language which can be evaluated semantically using the formal model in question. Along these lines, in Section 4 we develop the rudiments of a CNL interface to defeasible theories, which can be evaluated using the direct-stable-semantics.

Specifically, we motivate and show the design of an implementation of what co-authors of ours have dubbed the EMIL (acronym for “Extracting Meaning from Inconsistent Language”) pipeline [SWD]. The input of the pipeline in question are collections of rules in an extension of an existing CNL, ACE [FKK08], allowing for the expression of potentially conflicting defeasible rules in the form of normality assumptions in addition to strict rules. Such rules are, whenever possible, translated into defeasible theories and can, hence, be made sense of in terms of the direct stable semantics. Finally, stable

[7]`https://www.dbai.tuwien.ac.at/proj/adf/yadf/`

[8]See e.g. [CTO14] and particularly [WvEH16] for proposals in this direction.

sets (expressing different manners of resolving conflicts in the defeasible theories) are verbalised using ACE.

The design of the system we develop here is based on our experience with an initial experimental prototype implementing EMIL we developed relying heavily on an existing interface to rule systems for ACE, `ACERules` [Kuh07]. The nature of this initial prototype is sketched in Section 4.2.2. We encountered several issues with this system which led to an in-depth investigation of the inner workings of `ACERules`; the results of which forms a large part of Section 4. More to the point, in Section 4.1 we give a high level description of the system `ACERules`, paying attention not only to features we have adopted in our design but particularly to the sources of the problems we found. Note that such a description is unavailable to date; the only publications on `ACERules` [Kuh07, Kuh10] deal with so called grouping (see Section 4.1.3) and in a less detailed manner than we do.

The system `ACERules` basically works by filtering ACE texts corresponding to general rules allowing, for instance, existentially quantified variables as well as conjunctions of atoms in the heads and bodies of rules. In particular, these atoms can appear negated (either via strong negation or negation-as-failure). A large part of the system `ACERules` consists in several transformations attempting to make such general rules, which we call $\exists$-rules [GGLS15], conform to the format of the rule systems it provides an interface to; the most general of these are normal logic programs with strong as well as negation-as-failure [GL90].

The issues we encountered when using the system `ACERules` are with these transformations. Summarising, the problem is that in some cases the transformations introduce semantic errors in the interpretation of ACE texts. On the other hand, providing an interface to rule systems while preserving the semantics, leads to many natural ACE texts expressed as rules being filtered-out when they could be handled with modern-day rule systems.

In Section 4.2 we outline the already alluded to alternative design for an implementation of EMIL. The crucial aspect of this implementation is that we target $\exists$-rules, also including defeasible in addition to strict $\exists$-rules, rather than normal rules. Our translation of such $\exists$-rules to normal rules, which we further discuss as well as sketch in Section 4.2.4.1, allows us to circumvent the main problems we found with `ACERules` in a uniform manner. We show encodings for evaluating defeasible theories via the direct stable semantics using ASP in Section 4.2.4.2. In contrast to previous existing encodings [SW17], these encodings are optimised for defeasible theories with variables as well as function symbols (which we make use of in our translation of $\exists$-rules to normal rules).

In Section 4.3 we finally give an extended example of the use of the implementation of the EMIL pipeline in the context of ACEWiki [Kuh09], a prototye of an online encyclopedia expressed in ACE. In Section 4.4 we present the functioning of the system we implemented, `emil`[9], based on the ideas developed in Section 4 as well as the results of preliminary experiments.

[9] `https://www.dbai.tuwien.ac.at/proj/grappa/emil/`

Before presenting our contributions in Section 2 we present the formal background required to follow our work. We assume, on the other hand, an understanding of basic notions and notation of classical propositional as well as first order logic.

Publications Parts of the results in this thesis have been published or submitted for publication. Our complexity-sensitive QBF encodings w.r.t. the stable semantics for ADFs are a part of the following publication (which otherwise mainly includes results from [Dil14]):

- Martin Diller, Johannes Peter Wallner, and Stefan Woltran. Reasoning in abstract dialectical frameworks using quantified Boolean formulas. *Argument & Computation*, 6(2):149–177, 2015

We presented the results of our initial experiments in using different QBF solvers and preprocessors on our link-information-sensitive QBF encodings for ADFs at the 2018 "International Workshop on Quantified Boolean Formulas and Beyond" (QBF'2018) colocated with the Federated Logic Conference 2018 (FLoC) at Oxford, in the UK.

We first presented our work on dynamic ASP encodings for ADFs at AAAI'2017:

- Gerhard Brewka, Martin Diller, Georg Heissenberger, Thomas Linsbichler, and Stefan Woltran. Solving Advanced Argumentation Problems with Answer-Set Programming. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the 31rst Conference on Artificial Intelligence (AAAI 2017)*, pages 1077–1083. AAAI Press, 2017

We later extended this work by also adding encodings for the grounded and stable semantics as well as giving prototypical proofs in:

- Gerhard Brewka, Martin Diller, Georg Heissenberger, Thomas Linsbichler, and Stefan Woltran. Solving Advanced Argumentation Problems with Answer-Set Programming. Submitted to TPLP - Special issue on argumentation and logic programming - frontiers and relations.

Regarding our work on the implementation of the EMIL pipeline, our first publication is the result of a presentation at IWCS'17 (here we make use of the system built on top of `ACERules` to motivate the EMIL pipeline with an extended example of its use in the context of ACEWiki):

- Martin Diller, Adam Wyner, and Hannes Strass. Defeasible AceRules: A prototype. In Claire Gardent and Christian Retoré, editors, *Proceedings of the 12th International Conference on Computational Semantics (IWCS 2017)*, September 2017

Our ideas for a novel implementation of EMIL first appear sketched in:

- Hannes Strass, Adam Wyner, and Martin Diller. EMIL: Extracting Meaning from Inconsistent Language. Towards argumentation using a controlled natural language interface. Submitted to International Journal of Approximate Reasoning

The following submission is more focused on the design of our new implementation, while also providing more details on the motivations behind adopting a different strategy to the transformations implemented in the system `ACERules`:

- Martin Diller, Adam Wyner, and Hannes Strass. Making sense of conflicting (defeasible) rules in the controlled natural language ACE: design of a system with support for existential quantification using skolemization. In *Proceedings of the 13th International Conference on Computational Semantics (IWCS 2019)*, 2019. To appear

Further articles co-authored by the author of this work in the context of the PhD which, while being related to this work, do not contain results directly included here are listed below:

- Martin Diller, Wolfgang Dvořák, Jörg Pührer, Johannes Peter Wallner, and Stefan Woltran. Applications of ASP in Formal Argumentation. In *Proceedings of the 2nd Workshop on Trends and Applications of Answer Set Programming (TAASP 2018)*, 2018. Available online at `http://www.kr.tuwien.ac.at/events/taasp18/papers/TAASP_2018_paper_16.pdf`
- Martin Diller, Atefeh Keshavarzi Zafarghandi, Thomas Linsbichler, and Stefan Woltran. Investigating Subclasses of Abstract Dialectical Frameworks. In *Proceedings of the 7th International Conference on Computational Models of Argument COMMA 2018*, volume 305 of *Frontiers in Artificial Intelligence and Applications*, pages 61–72. IOS Press, 2018
- Martin Diller, Adrian Haret, Thomas Linsbichler, Stefan Rümmele, and Stefan Woltran. An Extension-Based Approach to Belief Revision in Abstract Argumentation. *Int. J. Approx. Reasoning*, 93:395–423, 2018
- Martin Diller and Anthony Hunter. Encoding monotonic multiset preferences using CI-nets. In Bernhard Mitschang, Norbert Ritter, Holger Schwarz, Meike Klettke, Andreas Thor, Oliver Kopp, and Matthias Wieland, editors, *Proceedings of the 1rst Workshop on Preferences and Personalisation in Informatics (PPI 2017) at the 17th International Conference on Database Systems for Business, Technology, and Web of the German Informatics Society (BTW 2017)*, volume P-266 of *LNI*, pages 169–180. GI, 2017

- Martin Diller and Anthony Hunter. Encoding monotonic multi-set preferences using CI-nets: preliminary report. *CoRR*, abs/1611.02885, 2016
- Martin Diller, Adrian Haret, Thomas Linsbichler, Stefan Rümmele, and Stefan Woltran. An Extension-Based Approach to Belief Revision in Abstract Argumentation. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 2926–2932. AAAI Press, 2015

CHAPTER 2

Formal background

In this chapter we present the formal background necessary to follow our work. Specifically, we detail the basics of quantified boolean formulas in Section 2.1, answer-set-programming in Section 2.2, abstract dialectical frameworks in Section 2.3, and defeasible theories with associated direct-stable-semantics in Section 2.4. An overview of the complexity classes we make reference to in this work is provided in Section 2.5. As already mentioned in the introduction, we assume basic familiarity with the notation and theory of classical propositional and first order logic; see e.g. [End72].

2.1 Quantified Boolean Formulas (QBFs)

Quantified boolean logic is an extension of propositional logic. As has already been hinted at in the introduction to this work, from a computational perspective, one of the main reasons behind the importance of quantified boolean logic is its connection with the complexity class PSPACE and the polynomial hierarchy (see Section 2.5 for an overview of the complexity classes referred to in this work). The satisfiability problem for quantified boolean formulas, QSAT, can be considered the "prototypical problem" for the class PSPACE in the sense that this class can be defined as the set of problems that can be expressed in terms of QSAT in an efficient manner. Moreover, there are subclasses of quantified boolean formulas or QBFs whose satisfiability problem is prototypical for each level of the polynomial hierarchy. For the theory of QBFs the standard overview is still [KB09].

As we also referred to in the introduction to this work, quantified boolean logic is also growing increasingly relevant to AI & computer science from a more practical perspective. The reason for this are the increasing efforts (and several significant accomplishments) in replicating the success of SAT solving in the development and application of QSAT solvers to hard computational problems. See e.g. reports on previous international competions for QSAT solvers as QBFEVAL or the QBF galleries [JJK+14, LSG16]; as well as the dedicated webpage for the results of the latest editions[1].

2.1.1 Syntax

The basis of propositional logic is a countable set of propositional variables $\mathcal{P}$, to which we also refer to as atoms. Propositional formulas are built recursively from the propositional atoms and the connectives $\wedge$, $\vee$, and $\neg$, denoting the logical conjunction, disjunction and negation, respectively. We also allow the use of the truth constant $\top$, allowing a direct representation of the truth value true. Quantified boolean formulas, or QBFs for short, additionally use the universal quantifier $\forall$ and the existential quantifier $\exists$.

Definition 1. *The set of quantified boolean formulas (QBFs) is defined inductively as follows:*

- *Any propositional variable $p \in \mathcal{P}$ is a QBF.*
- *The logical constant $\top$ is a QBF.*
- *If ϕ and ψ are QBFs, then so is $(\neg\phi)$, $(\phi \wedge \psi)$, and $(\phi \vee \psi)$.*
- *If $p \in \mathcal{P}$ is a propositional variable and ϕ is a QBF, then $(\exists p\phi)$ and $(\forall p\phi)$ are QBFs.*
- *Nothing else is a QBF.*

Any QBF without the quantifiers $\exists$ and $\forall$ is a propositional formula. We define shorthands for the truth constant false, (material) implication, equivalence, and exclusive or. Given QBFs ϕ and ψ, the respective definitions are as follows:

$$\begin{aligned}
\bot &:= \neg\top \\
\phi \rightarrow \psi &:= \neg\phi \vee \psi \\
\phi \leftrightarrow \psi &:= (\phi \rightarrow \psi) \wedge (\psi \rightarrow \phi) \\
\phi \oplus \psi &:= (\phi \vee \psi) \wedge \neg(\phi \wedge \psi)
\end{aligned}$$

We also define shorthands for n-ary conjunctions and disjunctions. Given a set $F = \{\phi_1, \phi_2, \ldots, \phi_n\}$ of QBFs, these abbreviations are defined as follows:

$$\bigvee_{\phi \in F} := \bigvee_{i=1}^{n} \phi_i := \phi_1 \vee \phi_2 \vee \ldots \vee \phi_n$$

[1]`http://www.qbflib.org/index_eval.php`

$$\bigwedge_{\phi \in F} := \bigwedge_{i=1}^{n} \phi_i := \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_n.$$

If $F = \emptyset$ (or $n = 0$), we stipulate the above abbreviations to simplify to $\bot$ for the empty disjunction and $\top$ for the empty conjunction.

For purposes of clarity, we will in this work often omit parentheses, for which we introduce a ranking to the logical symbols we have presented above. We stipulate that $\neg$, $\exists p$, and $\forall p$ have the same ranking. Also, these have a higher ranking than the connectives $\wedge$, $\vee$, $\oplus$, $\rightarrow$, and $\leftrightarrow$ which again have the same ranking. When parentheses are omitted, parentheses should be read into the formula according to this ranking (i.e. connectives higher in the ranking "bind stronger") to avoid ambiguity. We will further simplify our rendering of formulas by assuming that binary connectives associate to the left (so, for example, $\phi_1 \wedge \phi_2 \wedge \phi_3$ should be read as $((\phi_1 \wedge \phi_2) \wedge \phi_3)))$. Also, we will often not write outermost parentheses of QBFs.

We also introduce some simplifications for writing out QBFs with quantifiers. Specifically, if $P = \{p_1, p_2, \ldots, p_n\}$ is a set of propositional variables and ϕ a QBF, then $QP\phi$ and $Qp_1p_2 \ldots p_n\phi$ are to be read as $(Qp_1(Qp_2(\ldots(Qp_n(\phi)))))$ for any $Q \in \{\exists, \forall\}$. In particular, if $P = \emptyset$ then $QP\phi$ is to be read as ϕ. Also, we will sometimes use $QP_1 \ldots P_n\phi$ for sets of propositional variables $P_1, \ldots, P_n$ $(n > 1)$, as an abbreviation of $Q(P_1 \cup \ldots \cup P_n)\phi$. We call successive quantifiers of the same kind occurring in a certain formula a quantifier block.

We assume the notions of subformula of a QBF as well as occurrence of one formula (e.g. a propositional variable) in another to be clear. E.g. the subformulas of the QBF $\forall p \exists q(p \wedge q)$ are $\forall p \exists q(p \wedge q)$, $\exists q(p \wedge q)$, $(p \wedge q)$, p, and q. Any of the subformulas of $\forall p \exists q(p \wedge q)$ occurs in this QBF. Note that there may be multiple occurrences of a subformula in a QBF; when this is the case, it should be clear from context to which occurrence we are referring to and, hence, we do not define this concept formally here.

Further important syntactical notions regarding QBFs are the scope in which a quantifier is applied and whether a variable appears bound by a quantifier or not in such a formula.

Definition 2. *The scope of a quantifier $Q \in \{\forall, \exists\}$ in a QBF of the form $Qp\phi$ is the QBF ϕ. An occurrence of a variable p in a QBF ϕ is free if it does not occur in the scope of a quantifier in the QBF, otherwise the occurrence of p is bound. If a QBF ϕ contains no free variable occurrences, then ϕ is closed, otherwise ϕ is open. $FREE(\phi)$ denotes the set of free variables of a QBF ϕ.*

With these notions in hand, substitution of formulas for variables occurring in a QBF can be defined:

Definition 3. *Let ϕ be a QBF, $\{\psi_1, \ldots, \psi_n\}$ a set of QBFs such that none of the propositional variables in $P = \{p_1, \ldots, p_n\}$ occurs free in any of the ψ_is for $1 \leq i \leq n$.*

Then, $\phi[p_1/\psi_1, \ldots, p_n/\psi_n]$ denotes the QBF which results by uniform substitution of all free occurrences of the variables p_i in ϕ by the corresponding ψ_i for $1 \leq i \leq n$.

Rather than explicitly listing the substitutions as in the notation introduced in Definition 3, we will often describe them as in the alternative notation $\phi[p_i/\psi_i \mid 1 \leq i \leq n]$ (in this example, assuming p_i and ϕ_i have been defined for each $1 \leq i \leq n$).

2.1.2 Semantics

Semantics for QBFs are defined in terms of (two valued) interpretations of propositional variables, an interpretation being a mapping $v : \mathcal{P} \rightarrow \{\mathbf{t}, \mathbf{f}\}$ that defines for each propositional variable a truth assignment. Here **t** stands for "true" and **f** for "false". We will sometimes explicitly highlight that an interpretation v is defined on a set $P \subseteq \mathcal{P}$.

The extension of an interpretation on atoms to that of QBFs is given in Definition 4.

Definition 4. *Given a QBF ϕ and an interpretation v, then ϕ evaluates to true under v (v satisfies ϕ or v is a model of ϕ, denoted by $v \vDash \phi$ or, simply, $v(\phi) = \mathbf{t}$) if one of the following holds, with $p \in \mathcal{P}$ and QBFs ψ, ψ_1 and ψ_2.*

- *$\phi = p$ and $v(p) = \mathbf{t}$;*
- *$\phi = \top$;*
- *$\phi = \neg\psi$ and ψ does not evaluate to true under v;*
- *$\phi = \psi_1 \wedge \psi_2$ and both ψ_1 and ψ_2 evaluate to true under v;*
- *$\phi = \psi_1 \vee \psi_2$ and one of ψ_1 and ψ_2 evaluates to true under v;*
- *$\phi = \exists p\psi$ and one of $\psi[p/\top]$ and $\psi[p/\bot]$ evaluates to true under v;*
- *$\phi = \forall p\psi$ and both $\psi[p/\top]$ and $\psi[p/\bot]$ evaluate to true under v.*

We say that a a QBF ϕ evaluates to false under an interpretation v if it does not evaluate to true under v. We denote this as $v(\phi) = \mathbf{f}$ or also $v \nvDash \phi$. Now we have that a QBF ϕ is satisfiable if there is an interpretation v s.t. $v(\phi) = \mathbf{t}$. It is unsatisfiable otherwise. A QBF is valid if for every interpretation v, $v(\phi) = \mathbf{t}$. This is denoted as $\vDash \phi$. If ϕ is not valid -,i.e. there is an interpretation v s.t. $v(\phi) = \mathbf{f}$,- this is denoted as $\nvDash \phi$. For closed QBFs the notions of satisfiability and validity are equivalent; hence, closed QBFs that are satisfiable are often called "true" while those which are unsatisfiable are called "false".

Two QBFs ϕ_1 and ϕ_2 are logically equivalent whenever $v(\phi_1 \leftrightarrow \phi_2) = \mathbf{t}$ for every interpretation v. We write this as $\phi_1 \equiv \phi_2$. The truth value of a QBF is uniquely determined by the truth values of its components; this is the "principle of extensionality" (for QBFs). This fact has the important consequence that a subformula ψ of a QBF ϕ can be replaced for another formula ψ' that is logically equivalent to it (i.e. $\psi \equiv \psi'$), without altering the truth value of ϕ. The formal statement of this result is usually called the "replacement theorem" and allows syntactical transforming QBFs while preserving the semantics.

There are many well known semantic equivalences that are used for meaning preserving syntactic transformations of QBFs. Although we make extensive use of them, we don't list any of the (many) important semantic equivalences that are known for propositional logic and all generalise to QBFs, e.g. commutativity of $\wedge$ and $\vee$ or De Morgan's laws. We do conclude this subsection by listing some equivalences that are specific to QBFs and that are equally relevant to this work.

Proposition 1. *Let ϕ and ψ be QBFs with p not occurring free in ψ. Also let $Q \in \{\exists, \forall\}$, $\overline{Q} := \forall$ if $Q = \exists$, and $\overline{Q} := \exists$ if $Q = \forall$. The following equivalences then hold.*

- $(\neg Qp\phi) \equiv \overline{Q}p(\neg\phi)$,
- $(Qp\phi \wedge \psi) \equiv Qp(\phi \wedge \psi)$,
- $(\psi \wedge Qp\phi) \equiv Qp(\psi \wedge \phi)$,
- $(Qp\phi \vee \psi) \equiv Qp(\phi \vee \psi)$,
- $(\psi \vee Qp\phi) \equiv Qp(\psi \vee \phi)$.

Note that a consequence of the definition of the connective "$\rightarrow$" and Proposition 1 is that also the following equivalences which we make extensive use of hold:

- $(Qp\phi \rightarrow \psi) \equiv \overline{Q}p(\phi \rightarrow \psi)$,
- $(\psi \rightarrow Qp\phi) \equiv Qp(\psi \rightarrow \phi)$.

Here again $Q \in \{\exists, \forall\}$ and p does not occur free in ψ.

2.1.3 Relating syntax and semantics

The syntax and semantics of QBFs are closely related. We now state some elemental lemmas which we make repeated use of in the proofs of Section 3.1 of this work and which allow us to switch from writing about one to writing about the other.

We start of by defining a form of "substitution" for interpretations. Given two interpretations v and w as well as a set of propositional variables P, $v[P/w(P)]$ denotes the interpretation v' defined as:

- $v'(p) := w(p)$ for $p \in P$.
- $v'(q) := v(q)$ if $q \in \mathcal{P}$ and $q \notin P$.

We will often use the simplified notation $v[p/x]$ with $x \in \{\mathbf{t}, \mathbf{f}\}$ in the case that $P = \{p\}$ and $w(p) = x$ in the definition above.

Note that we use the same notational device - ,brackets, - both for substitution for formulas as well as for substitution for interpretations. We trust that the place where the brackets appear makes it clear to the reader whether we use brackets denoting the first or the latter.

The following lemma and corollary are straightforward consequences of the semantics of QBFs.

Lemma 1. *Let ψ be a QBF, and P a set of propositional variables. If $\phi = \exists P\psi$ ($\phi = \forall P\psi$), then $v \models \phi$ if and only if $v[P/w(P)] \models \psi$ for some (all) interpretation(s) w.*

Corollary 1. *Let ψ be a closed QBF, and P a set of propositional variables. If $\phi = \exists P\psi$ ($\phi = \forall P\psi$), then ϕ is true if and only if for some (all) interpretation(s) v, $v \models \psi$.*

One final use of brackets as notational device we introduce for QBFs is $\phi[P/v(P)]$ to denote the formula ϕ where every occurrence of any variable $p \in P$ is replaced for $\top$ in case that $v(p) = \mathbf{t}$ and replaced for $\bot$ in the case that $v(p) = \mathbf{f}$. For $\phi[\{p\}/v(\{p\})]$ we use the simplified notation $\phi[p/v(p)]$. Another immediate result relating syntax and semantics that we make repeated use of in this work is then the following.

Lemma 2. *Let v be an interpretation, ϕ a QBF, and $p \in \mathcal{P}$. Then $v \models \phi$ if and only if $v \models \phi[p/v(p)]$. Also, if P are all the free variables of ϕ, then $v \models \phi$ if and only if for all interpretations w it is the case that $w \models \phi[P/v(P)]$.*

2.1.4 Prenex normal forms and complexity

For computational purposes it is often useful to restrict attention on formulas with a specific syntactic structure. Ideally, such a restriction should still allow to express all possible QBFs. The prenex normal form as well as prenex conjunctive normal, are two such normal forms. The first is particularly important since there is a one to one correspondence between the complexity of reasoning for certain types of QBFs in prenex normal form and the different levels of the polynomial hierarchy. The prenex conjunctive normal form on the other hand imposes a further restriction on the prenex normal form and generalises the conjunctive normal form for propositional formulas to QBFs. Its importance is due to the fact that most QBF solvers expect the input QBFs to be in a variant of this normal form (the QDIMACS format).

We first remind the reader that a clause is a disjunction of literals, literals being propositonal atoms or negations thereof. I.e. a literal is of the form p or $\neg p$ with $p \in \mathcal{P}$, the first being called a positive literal, while the second is a negative literal. A formula is in conjunctive normal form (CNF) if it is a conjunction of clauses. Formulas in conjunctive normal form capture all propositional formulas, i.e. given a propositional formula ϕ there is a formula ϕ' in CNF such that $\phi \equiv \phi'$.

It is well known, nevertheless, that transformation of an arbitrary propositional formula to CNF can lead to an exponential explosion. A linear transformation to CNF form is the so called "Tseitin encoding" first defined in [Tse68] which works by replacing subformulas with new variables and then adding clauses to the original formula specifying the relationship between the newly introduced variables and the subformulas. This results in an equisatisfiable formula, i.e. if ϕ is the original formula and ϕ' is the one obtained via the Tseitin transformation then ϕ is satisfiable if and only if ϕ' is satisfiable. When satisfiability of the original formula is the issue of concern, as will mostly be the case in this work, this is obviously not a disadvantage.

To introduce the notion of a QBF in prenex conjunctive normal form we first need to introduce the notion of a QBF ϕ being "standarized apart". This is the case when the following conditions hold:

- no variable occurring in ϕ occurs both free and bound,
- for each $Q_1, Q_2 \in \{\exists, \forall\}$, if $Q_1 p$ and $Q_2 q$ are two distinct occurrences of quantifiers in ϕ, then p and q are distinct variables.
- For each subformula $Qp\psi$ with $Q \in \{\exists, \forall\}$ occurring in ϕ, p is a free variable in ψ.

An arbitrary QBF ϕ can be transformed in linear time into a ϕ' which is equivalent to ϕ and standarized apart by renaming variables which do not satisfy the first two conditions above and removing Qp for $Q \in \{\exists, \forall\}$ in $Qp\psi$ when p does not occur freely in ψ. That the resulting formula ϕ' is equivalent to ϕ is based on the fact that for a propositional variable p not occurring freely in a QBF ψ and $Q \in \{\exists, \forall\}$, $(Qq\psi) \equiv (Qp\psi[q/p])$ and $(Qp\psi) \equiv (\psi)$.

Now the definition of the prenex normal form as well as the prenex conjunctive normal form can be given as follows:

Definition 5. *A QBF ϕ is in prenex normal form (PNF) if it is standarized apart and it is of the form $Q_1P_1Q_2P_2 \dots Q_nP_n\psi$ where ψ is a propositional formula, $Q_i \in \{\exists, \forall\}$ for $1 \leq i \leq n$, and the P_is are (mutually disjoint) sets of propositional variables. ψ is called the matrix of ϕ and $Q_1P_1Q_2P_2 \dots Q_nP_n$ is the prefix of ϕ. Finally, ϕ is in prenex conjunctive normal form (PCNF) if it is in prenex normal form and its matrix is in conjunctive normal form.*

Just as CNFs capture propositional formulas, there is an equivalent formula in PNF as well as PCNF for every QBF. A standard procedure for transforming a QBF ϕ into an equivalent formula ϕ' in PNF is by first standarizing apart. Then, defined connectives are eliminated in terms of their definitions and subformulas of the form $\neg\forall p\psi$ are transformed into $\exists p\neg\psi$ and $\neg\exists p\psi$ into $\forall p\neg\psi$ in recursive manner. Subsequently, quantifiers are "pulled out" by using the last four equivalences in Proposition 1 (from left to right).

For transforming a QBF in PNF to PCNF, the matrix of the formula in prenex normal form can be transformed into CNF. As already indicated, this last step can lead to an exponential explosion, but the Tseitin procedure can be adapted to QBFs. Using the (adapted) Tseitin procedure it is possible to achieve a linear transformation resulting in an equivalent (since closed) formula.

The computational complexity of deciding whether a QBF ϕ in PNF is satisfiable depends on the prefix type. Every propositional formula has the prefix type $\Sigma_0 = \Pi_0$. Let ϕ be a closed QBF with prefix type Σ_i (respectively, Π_i) and P a set of $m > 0$ propositional variables. Then the formula $\forall P\phi$ (respectively $\exists P\phi$) is of type Π_{i+1} (respectively Σ_{i+1}) for $i \geq 0$. Now deciding whether a QBF ϕ in PNF is satisfiable is Σ_k^P complete if ϕ has prefix type Σ_k and otherwise if ϕ has prefix type Π_k, then the problem is Π_k^P complete ($k \geq 1$) [Sto76, Wra76] (see Section 2.5 for the definitions of the complexity classes Σ_k^P and Π_k^P for every $k \geq 1$). Deciding whether an arbitrary QBF (i.e. not in any normal form) is satisfiable is PSPACE-complete in general [SM73].

2.2 Answer Set Programming (ASP)

As indicated in the introduction to this work, answer set programming or ASP for short, is a paradigm for declarative problem solving (first propounded in [MT99, Nie99]). Roughly, in answer set programming search or optimisation problems are encoded using high level logical rules which by means of a grounding procedure are turned in a more low level propositional like format that is used by solvers to compute (optimal) solutions to the problems of interest [SW18a]. The rules at the start of the answer set programming workflow form “logical programs”, the solutions being the “answer sets”.

Having its roots in the development of the stable semantics for logic programs [GL88, GL90, GL91] (see [MNT11] for an early historical overview also detailing influences from knowledge representation and database theory), a large part of ASPs current success lies in its increasingly rich high level modeling language [GS16], even being Turing complete (i.e. as powerful as any programming language) in general (e.g. when function symbols are allowed [AFL10]). Another reason for the success of ASP is the flourishing of practical techniques and methodologies for ASP, starting with the generate & test (or guess & check) methodology that is at the heart of the ASP paradigm [JN16]. Here parts of a program delineate candidates for a solution to a problem and other parts (the “constraints”) check whether the candidates are indeed solutions. Finally, powerful systems (e.g. `DLV` [AAC+18] and the set of ASP tools developed at the University of Potsdam [GKK+18]) exist today; these automate both the grounding of the logical programs as well as the search for the answer sets.

In this section we mainly introduce the syntax and semantics of logical programs as we make use of in this work (sections 2.2.1 and 2.2.2). This corresponds to a powerful yet nevertheless restricted subset of the modeling languages supported by the most important of todays existing ASP solvers. We refrain from attempting to explain here all the intricacies of the ASP problem solving approach, but will rather spell out adopted ASP techniques such as the generate & test methodology, the saturation technique [EG95], and encoding of existentially quantified variables by making use of function symbols [GGLS15] in the relevant sections of this work. We refer to recent special editions of important AI journals [BET16, SW18b] for introductions on all aspects of the ASP paradigm from the foundations to grounding and solving techniques. There are also excellent older short survey articles [BET11, EIK09] as well as a newer entire book on practical ASP [GKKS12] available.

2.2.1 Syntax

From a formal perspective logic programs can be seen as being built from atoms and terms constructed on the basis of a language $\mathcal{L} = (\mathcal{V}, \mathcal{C}, \mathcal{F}, \mathcal{P})$. Specifically, the language $\mathcal{L}$ consists of disjoint countable sets of variables $\mathcal{V}$, constant symbols $\mathcal{C}$, function symbols $\mathcal{F}$, and predicate symbols $\mathcal{P}$. Function and predicate symbols have associated arities which are functions $\alpha^{\mathcal{F}} : \mathcal{F} \mapsto \mathbb{N}_{>0}$ and $\alpha^{\mathcal{P}} : \mathcal{P} \mapsto \mathbb{N}$ mapping function and predicate symbols respectively to natural numbers (i.e. the numbers $0, 1, 2, 3, \ldots$). It is a common convention that we will follow in this work to write constants, functions, and predicates as strings starting with lower case letters and variables as strings starting with upper case letters. Moreover, constants may be integers.

The set of terms of $\mathcal{L}$, $\mathcal{T}(\mathcal{L})$ are defined recursively as follows:

- If $v \in \mathcal{V}$ then $v \in \mathcal{T}(\mathcal{L})$;
- if $c \in \mathcal{C}$ then $c \in \mathcal{T}(\mathcal{L})$;
- if $f \in \mathcal{F}$, $\alpha^{\mathcal{F}}(f) = n$ with $n > 0$, and $t_i \in \mathcal{T}(\mathcal{L})$ for $1 \leq i \leq n$, then also $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{L})$;
- nothing else is in $\mathcal{T}(\mathcal{L})$.

The ground terms $\mathcal{GT}(\mathcal{L})$ are the terms with no occurrence of variables.

The set of atoms $\mathcal{A}(\mathcal{L})$ are defined making use of the terms as well as the predicate symbols as detailed next:

- If $a \in \mathcal{P}$ and $\alpha^{\mathcal{P}}(a) = 0$, then $a \in \mathcal{A}(\mathcal{L})$;
- if $a \in \mathcal{P}$, $\alpha^{\mathcal{P}}(a) = n$ with $n > 0$, and $t_i \in \mathcal{T}(\mathcal{L})$ for $1 \leq i \leq n$, then $p(t_1, \ldots, t_n) \in \mathcal{A}(\mathcal{L})$;
- nothing else is in $\mathcal{A}(\mathcal{L})$.

The ground atoms $\mathcal{GA}(\mathcal{L})$ are the atoms built using terms from $\mathcal{GT}(\mathcal{L})$ only (i.e. without occurrence of variables).

Logical programs, or simply programs for short, are sets of rules. A (disjunctive) rule r with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$ is of the form

$$a_1 \vee \ldots \vee a_n \text{ :- } b_1, \ldots, b_k,\ \textit{not}\ b_{k+1}, ...,\ \textit{not}\ b_m$$

where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms, and "*not*" stands for default negation or negation-as-failure. An atom a is a positive literal, while *not* a is a default negated literal. The head of r is the set $H(r) := \{a_1, \ldots, a_n\}$ and the body of r is $B(r) := B^+(r) \cup B^-(r)$. $B^+(r) = \{b_1, \ldots, b_k\}$ is the positive body of r, while $B^-(r) = \{b_{k+1}, \ldots, b_m\}$ is the negative body. We will often represent bodies of rules or even just parts of bodies of rules as sets of atoms rather than lists of atoms (atoms separated by a comma). Informally, a rule r is a justification to "establish" or "derive" that at least one of the atoms $a_1, \ldots, a_n$ in the head is true, if all literals of the body of r are true in the following sense: a non-negated literal b_i is true if it has a derivation, while a negated one, *not* b_j, is true if the atom b_j does not have a derivation [BET11].

The rule r is normal (or disjunction free) if $n \leq 1$ and a constraint if $n = 0$. The rule r is safe if each variable in r occurs in $B^+(r)$. We only allow safe rules in programs. The rule r is function free if no function symbols occur in r. Moreover, the rule r is ground if no variable occurs in r, i.e. for all atoms a in r it is the case that $a \in \mathcal{GA}(\mathcal{L})$.

A fact is a ground disjunction free rule with an empty body. If each rule in the program is function free and normal, we call the program normal. If each rule in the program is function free, then the program is function free. In the same manner, if each rule in the program is ground, the program is a ground program.

2.2.2 Semantics

As our brief description of the ASP paradigm at the beginning of this section suggests, the semantics of logical programs is typically defined in terms of the semantics of ground programs. For any program π , let $U_\pi \subseteq \mathcal{GT}(\mathcal{L})$ be the set of all ground terms that can be constructed using constants and function symbols appearing in π , while $B_\pi \subseteq \mathcal{GA}(\mathcal{L})$ is the set of all ground atoms that can be constructed from predicates in π with elements of U_π. Also, $Gr(\pi)$ is the set of rules σr obtained by applying, to each rule $r \in \pi$, all possible substitutions σ from the variables in r to elements of U_π. The latter are called the ground rules or groundings of r (with respect to π).

An interpretation for a logical program π is a set $I \subseteq B_\pi$. Such an interpretation I satisfies a ground rule r iff $H(r) \cap I \neq \emptyset$ whenever $B^+(r) \subseteq I$ and $B^-(r) \cap I = \emptyset$. A non ground rule r on the other hand, is satisfied by an interpretation I, iff I satisfies all groundings of r. As expected, an interpretation I satisfies a program π, if each $r \in \pi$ is satisfied by I.

Now, an interpretation $I \subseteq B_\pi$ is an answer set of the logical program π iff it is a subset minimal set satisfying the Gelfond-Lifschitz reduct

$$\pi^I = \{H(r) \text{ :- } B^+(r) \mid I \cap B^-(r) = \emptyset, r \in Gr(\pi)\},$$

i.e. the program obtained by essentially deleting from $Gr(\pi)$ all the rules r such that some body literal in $B^-(r)$ is in I . We denote the answer sets of π by $\mathcal{AS}(\pi)$.

Intuitively, an interpretation $I \subseteq B_\pi$ can be seen as an assumption about which negated literals are true and which are false; the reduct π^I then incorporates these assumptions. The fact that I satisfies π^I is then an indicator that I is not contradictory or “stable” w.r.t. π^I. Finally, minimality is a further constraint imposed in order to avoid interpretations which convey more information than that what is strictly required by the program π^I [EIK09].

2.2.3 Arithmetic functions and comparison predicates

As hinted at previously, current ASP systems have many modelling resources beyond those defined as part of the language in sections 2.2.1 and 2.2.2 (see [GS16] for an overview). In this work we make use of a subset of two of these: arithmetic functions and comparison predicates. Concretely, we make use of the arithmetic functions `&` (bitwise AND), `?` (bitwise OR), `-` (substraction), `+` (addition) and the comparison predicates for equality $=$ and disequality $\neq$. From a syntactic point of view these are simply special reserved function and predicate symbols respectively. The arity associated to all of these symbols is two. They are usually written in infix rather than prefix notation.

Semantics of logical programs having arithmetic functions and comparison predicates are defined as for logical programs without these special function and predicate symbols. The only differences are, first of all, that arithmetic functions are evaluated when grounding rules. Thus, for example, possible groundings of the rule r_1

$$result(X\&Y) \text{ :- } num(X), num(Y).$$

are the rules

$$result(1) \text{ :- } num(0), num(1).$$

and

$$result(0) \text{ :- } num(0), num(0).$$

(assuming 0 and 1 are part of the constants appearing in the program of which r_1 is part). Groundings of rules with arithmetic functions are defined only for substitutions that are meaningful for the arithmetic functions the rules contain. Thus, for instance, τr_1 for the substitution $\tau(X) = adam$ and $\tau(Y) = eve$ is not a valid grounding since *adam* and *eve* are not constants representing binary integers.

Grounding removes arithmetic functions; extending the notion of containment or satisfaction of a ground atom a by an interpretation I (i.e. the notion $a \in I$) for the case that a is a comparison predicate is the last step enabling the semantics of logical programs as defined in Section 2.2.2 to be applicable also to logical programs with arithmetic functions and comparison predicates. And this is simple to do: we say that $a \in I$ for a ground comparison predicate a when the comparison stated by a holds (in the "standard" interpretation of the comparison predicates). E.g. $1 = 1 \in I$ and $eve = eve \in I$, while $1 = 2 \notin I$ for any interpretation I. On the other hand, $1 \neq 2 \in I$, while $eve \neq eve \notin I$. Note thus that the same ground comparison predicates are true for all interpretations.

2.2.4 Reasoning and complexity

The main decisions problem for answer-set-programming are first of all, that of deciding whether a program has an answer-set: the existence problem. Also, given a program π and atom a deciding whether there is a $I \in \mathcal{AS}(\pi)$ for which $a \in I$. In analogous manner, deciding whether $a \in I$ for every $I \in \mathcal{AS}(\pi)$. The latter are the problem of credulous reasoning (or acceptance) and skeptical reasoning (or acceptance) respectively.

The existence problem is NEXPTIME$^{\text{NP}}$-complete for arbitrary ASP programs without function symbols [EGM97], while for programs with function symbols it is undecidable [AFL10]. The data-complexity (i.e. only the facts of the program change) of normal programs is NP-complete, while the existence problem for programs with disjunction is Σ_2^P-complete [EG95]. The combined complexity (i.e. also the rules of the program change) of normal ASP programs with predicates of bounded arity (there is a $k \geq 0$ s.t. such that the arity of every predicate in the program is less than k) is Σ_2^P-complete, while for disjunctive ASP programs (with predicates of bounded arity) it is Σ_3^P-complete [EFFW07]. From these results the complexity of credulous and skeptical reasoning can also be derived; e.g. and central for our work is that credulous reasoning for normal programs with bounded predicate size is Σ_2^P-complete and skeptical reasoning Π_2^P-complete. For disjunctive ASP programs of bounded predicate size credulous reasoning is Σ_3^P-complete and skeptical reasoning Π_3^P-complete. See Section 2.5 for the definition of the complexity classes.

2.3 Abstract Dialectical Frameworks (ADFs)

As indicated in the introduction to this work, Abstract Dialectical Frameworks or ADFs for short, have been introduced as an attempt to unify several existing generalizations of Dung's Argumentation Frameworks (AFs) in a principled manner. Despite their relatively young age, there has been quite a substantial amount of theoretical development around ADFs. This includes the development of several semantics specific to ADFs (e.g. [AF15]), alternative approaches to define ADF semantics (e.g. [Pol14]), as well as generalizations of ADFs (e.g. [PD14] and, more recently, [BSWW18]). In this work we follow the more usual definition of ADFs and their semantics as introduced by [BW10] (and refined in [BES+13]). We refer to [BES+18] for a quite recent more thorough discussion of the motivations and current theory built around ADFs, in particular also for an account of the semantics of ADFs (and their relationship with those of AFs) given in terms of approximation fixpoint theory. The latter has been fleshed out in detail mainly in [Str13].

2.3.1 Syntax

We start of with the definition of ADFs that more clearly shows the connections of ADFs with Dung's AFs and was proposed in [BW10]. Here ADFs are directed graphs whose vertices represent statements, positions or arguments. The links represent dependencies: the acceptance status of a node s only depends on the acceptance status of its parents (denoted $par_D(s)$)), that is, the nodes with a direct link to s. In addition, each node s has an associated acceptance condition C_s specifying the conditions under which s can be accepted. C_s is a function assigning to each subset of $par_D(s)$ one of the truth values **t**, **f**.

Definition 6. *An abstract dialectical framework (ADF) is a tuple $D = (S, L, C)$ where*

- *S is a set of statements (positions, arguments, nodes),*
- *$L \subseteq S \times S$ is a set of links,*
- *$C = \{C_s\}_{s \in S}$ is a set of total functions $C_s : 2^{par_D(s)} \to \{\boldsymbol{t}, \boldsymbol{f}\}$, one for each statement s. C_s is called the acceptance condition of s.*

We will most often use an alternative more economical way of representing ADFs. Here the acceptance conditions are expressed as propositional formulas. The links are then given implicitly by the atoms occurring in the acceptance conditions, i.e. whenever a $s' \in S$ occurs in the acceptance condition of $s \in S$ this means that the acceptance status of s is linked to that of s'.

Definition 7. *An abstract dialectical framework (ADF) is a tuple $D = (S, \{\phi_s\}_{s \in S})$ where S is the set of statements and each $s \in S$ has a propositional formula ϕ_s, the acceptance condition of s, associated to it.*

Although we do not make explicit the links of an ADF in the representation introduced in Definition 7, we will still often refer to the link from a statement s to a statement t by which we mean that s appears in the acceptance condition of t. Formally, given an ADF $D = (S, \{\phi_s\}_{s \in S})$, we define $L_D := \{(s, t) \mid s \textit{ occurs in } \phi_t\}$.

2.3.2 Semantics

A semantics σ assigns to an ADF D a collection of two or three valued interpretations over the statements in the ADF, denoted by $\sigma(D)$. The interpretations map statements to truth values. We use $\{\mathbf{t}, \mathbf{f}, \mathbf{u}\}$ as truth values, denoting true, false and undecided respectively. Three valued interpretations map statements to one of any of the latter truth values, while two valued interpretations use only the truth values $\mathbf{t}$ and $\mathbf{f}$.

The three truth values are partially ordered by $\leq_i$ according to their information content: we have $\mathbf{u} <_i \mathbf{t}$ and $\mathbf{u} <_i \mathbf{f}$ and no other pair in $<_i$. Then for truth values x and y, $x \leq_i y$ iff $x <_i y$ or $x = y$. The information ordering $\leq_i$ extends in a straightforward way to interpretations v_1, v_2 over a set of statements S in that $v_1 \leq_i v_2$ iff $v_1(s) \leq_i v_2(s)$ for all $s \in S$.

For a three valued interpretation v, we say that a two valued interpretation w extends or completes v iff $v \leq_i w$. This means that all statements mapped to $\mathbf{u}$ by v are mapped to $\mathbf{t}$ or $\mathbf{f}$ by w. We will thus say that the two valued interpretation w is a completion of v. The set of all completions of v are denoted by $[v]_2$.

When evaluating an acceptance condition of a statement s of an ADF D by a completion we will often treat the completion as if it were defined only on the parents of s in D, although strictly speaking completions are always defined on all statements of ADFs. The reason is that for purposes of evaluating an acceptance condition all that matters is the assignments the completion gives to the statements appearing in the acceptance condition.

Semantics of ADFs can be defined in terms of a characteristic operator, which generalizes the characteristic function often used to define the semantics of Dung's AFs [Dun95]. For an ADF $D = (S, \{\phi_s\}_{s \in S})$, $s \in S$ and a three valued interpretation v, the characteristic operator $\Gamma_D(v) := v'$ is given by

$$v'(s) := \begin{cases} \mathbf{t} \text{ if } w(\phi_s) = \mathbf{t} \text{ for all } w \in [v]_2 \\ \mathbf{f} \text{ if } w(\phi_s) = \mathbf{f} \text{ for all } w \in [v]_2 \\ \mathbf{u} \text{ otherwise} \end{cases}$$

That is, the operator returns a three valued interpretation, mapping a statement s to $\mathbf{t}$, or respectively $\mathbf{f}$, if all two-valued interpretations extending v evaluate ϕ_s to true, respectively false. If there are $w_1, w_2 \in [v]_2$, s.t. $w_1(\phi_s) = \mathbf{t}$ and $w_2(\phi_s) = \mathbf{f}$, then the result is $\mathbf{u}$. Note that the characteristic function is defined on three-valued interpretations, while we evaluate acceptance conditions under two-valued interpretations (two-valued completions of three-valued interpretations).

In this work we consider all of the main semantics for ADFs, i.e. the generalizations of the semantics for Dung's AFs defined in [Dun95]. These can be classified in two types, according to whether they return three valued interpretations or two-valued interpretations.

For those returning three valued interpretations we consider the admissible, complete, grounded, and preferred semantics denoted as *adm*, *com*, *grd*, *prf* respectively.

Definition 8. *Let D be an ADF. A three valued interpretation v is*

- *in $adm(D)$ if $v \leq_i \Gamma_D(v)$;*
- *in $com(D)$ if $v = \Gamma_D(v)$;*
- *in $grd(D)$ if $v \in com(D)$ and there is no $w \in com(D)$ with $w <_i v$;*
- *in $prf(D)$ if $v \in adm(D)$ and there is no $w \in adm(D)$ with $v <_i w$.*

No other elements except those stipulated by the items above are in $adm(D)$, $com(D)$, $grd(D)$ and $prf(D)$ respectively.

For any ADF one has that all preferred interpretations are complete and all complete interpretations are admissible. In [BES$^+$13] it is also shown that the operator Γ_D is monotonic for any ADF D. As a consequence, by the result in [Tar55], an interpretation v is the grounded interpretation of an ADF D if and only if it is equal to the $\leq_i$ least fixpoint (l.f.p.) of Γ_D. The least fixpoint can be calculated by iteratively applying Γ_D starting with the interpretation $\mathcal{U}^S$ mapping all statements $s \in S$ of D to $\mathbf{u}$. Then $\Gamma_D^0 := \mathcal{U}^S$ and $\Gamma_D^{i+1} := \Gamma_D(\Gamma_D^i)$ for $i \geq 0$.

The model (*mod*) and stable (*stb*) semantics return two valued interpretations. A two valued interpretation v is a model of an ADF $D = (S, \{\phi_s\}_{s\in S})$ if $v(s) = v(\phi_s)$ for all $s \in S$. The definition of stable model semantics for ADFs is inspired by the stable semantics for logic programs (see Section 2.2), its purpose being to disallow cyclic support within a model. It is defined via the notion of a reduct that resembles the reduct used to evaluate logic programs.

Definition 9. *Let $D = (S, \{\phi_s\}_{s\in S})$ be an ADF. A two-valued model v of D is a stable model of D iff $E_v = \{s \in S \mid v(s) = \boldsymbol{t}\}$ equals the statements set to true in the grounded interpretation of the reduced ADF $D^v = (E_v, \{\phi_s^v\}_{s\in E_v})$, where for $s \in E_v$ we set $\phi_s^v := \phi_s[b/\bot : v(b) = \boldsymbol{f}]$. If $v \downharpoonleft_{E_v}$ is the interpretation v projected on E_v, i.e. $v \downharpoonleft_{E_v} (s) = v(s)$ for $s \in E_v$ and undefined otherwise, then the latter can equivalently be expressed as $v \downharpoonleft_{E_v} \in grd(D^v)$.*

Note that by definition stable models are models, and since the only completion of a two valued interpretation is itself, it is also easy to see that every model is a preferred interpretation. We thus have the relationships between the semantics for ADFs depicted in Figure 2.1.

Example 1. *In Figure 2.2 we see an example ADF $D = (\{a, b, c\}, C)$ with the acceptance conditions C given by $\varphi_a = b \vee \neg b$, $\varphi_b = b$ and $\varphi_c = c \to b$. The acceptance conditions are shown below the statements in the figure.*

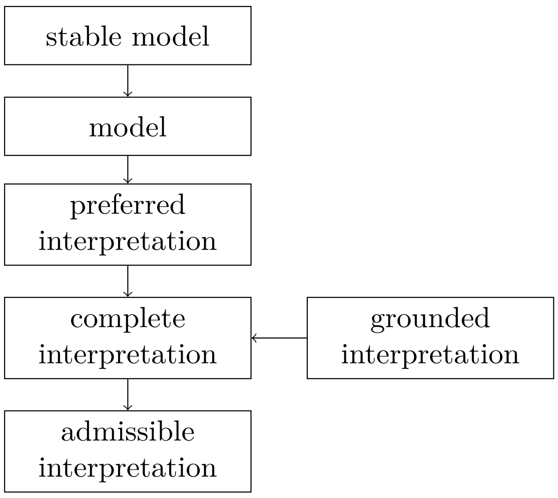

Figure 2.1: Relations between ADF semantics. An arrow from a semantics σ to another semantics σ' denotes that every σ-interpretation for an ADF D is also a σ' interpretation of D.

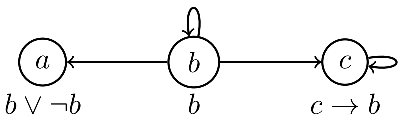

Figure 2.2: ADF example

The admissible interpretations of D are shown in Table 2.1. Furthermore the right-most column shows further semantics the interpretations belong to. For instance the interpretation v_8 mapping each statement to true is admissible, complete and preferred in D and a model of D. This ADF has no stable models. The only model of D is v_8, with the reduct of this model being $D^{v_8} = D$. The grounded interpretation of D is v_6, which is different than v_8. Therefore v_8 is not a stable model.

2.3.3 Reasoning and complexity

There are several reasoning tasks that can be defined on ADFs. Several of these are decision problems, i.e. problems with a yes/no answer. Some of the main decision problems, for which we develop implementation strategies in this chapter, are the following:

- $Ver_\sigma(D, v)$: Given an ADF D, a semantic σ, and an interpretation v, decide whether $v \in \sigma(D)$.
- $Cred_\sigma(D, s)$: Given and ADF D and a statement s of D, decide whether there exists a $v \in \sigma(D)$ s.t. $v(s) = \mathbf{t}$.

	a	b	c	
v_1	**u**	**u**	**u**	*adm*
v_2	**u**	**f**	**u**	*adm*
v_3	**u**	**t**	**u**	*adm*
v_4	**t**	**t**	**u**	*adm*
v_5	**u**	**t**	**t**	*adm*
v_6	**t**	**u**	**u**	*adm*, *com*, *grd*
v_7	**t**	**f**	**u**	*adm*, *com*, *prf*
v_8	**t**	**t**	**t**	*adm*, *com*, *prf*, *mod*

Table 2.1: All admissible interpretations of the ADF from Figure 2.2. The right most column shows further semantics the interpretations belong to.

σ	*adm*	*com*	*prf*	*grd*	*mod*	*stb*
$Cred_\sigma$	Σ_2^P-c	Σ_2^P-c	Σ_2^P-c	coNP-c	NP-c	Σ_2^P-c
$Skept_\sigma$	trivial	coNP-c	Π_3^P-c	coNP-c	coNP-c	Π_2^P-c
Ver_σ	coNP-c	DP-c	Π_2^P-c	DP-c	in P	coNP-c

Table 2.2: Complexity results for semantics of ADFs. From [SW15].

- $Skept_\sigma(D, s)$: Given and ADF D and a statement s of D, decide whether for all $v \in \sigma(D)$ it is the case that $v(s) = \mathbf{t}$.

The computational complexity of reasoning in ADFs is summarized in Table 2.2 (see Section 2.5 for an overview of the complexity classes relevant to this work). The results are from [BES+13] and [SW15].

Example 2. *Continuing Example 1 we can see that a is skeptically accepted w.r.t. preferred semantics in the ADF D. (i.e. $Skept_{prf}(a, D)$ is true). The statement b is not skeptically accepted for the preferred semantics, however it is credulously accepted under this semantics. In fact here all statements are credulously accepted under the admissible, complete, preferred and model semantics. As on all ADFs, credulous and skeptical acceptance coincide for the grounded semantics and in this example only a is accepted w.r.t. the grounded semantics.*

We will also in passing by in this work develop implementation strategies for the enumeration problem $Enum_\sigma(D)$, i.e. the problem of computing $\sigma(D)$ for an ADF D and a semantics σ.

To conclude this section, we define Dung's AFs (see [Dun95]) as specific instances of ADFs. In their original definition by Dung, AFs are directed graphs where nodes are interpreted as arguments and links represent attacks between arguments. AFs can be captured in a straightforward manner by ADFs as is also expressed in Definition 10.

Definition 10. *An argumentation framework (AF) is a pair $F = (A, R)$ where A is a set of arguments and $R \subseteq A \times A$ is a set of attacks. Given such an AF F, its associated ADF is $D^F := (A, \{\phi_a\}_{a \in A})$ with $\phi_a := \bigwedge_{b \in A,(b,a) \in R} \neg b$.*

ADFs generalize AFs in the specific sense that given an AF F and any semantics $\sigma \in \{adm, com, prf, stb\}$ (where the semantics for AFs are defined as in [Dun95]) one has that $\sigma(F) = \sigma(D^F)$. Moreover, $mod(D^F) = stb(D^F)$ for any ADF D which is associated to an AF F. The reason for the latter is that ADFs associated to AFs lack support links (see Definition 16 in Section 3.1.3.1) and, hence, the issue of circular supports that the stable semantic is designed to avoid does not arise for them. As a consequence, models as well as stable models for ADFs are proper generalizations of the stable semantics for AFs as defined by Dung.

2.4 Direct stable semantics for defeasible theories

The direct stable semantics for defeasible theories is the outcome of previous investigations [WBDC15, Str18] on defining a semantics for knowledge bases consisting of propositional strict and defeasible rules via abstract argumentation. The authors in [SW17] rather opt for an approach, where arguments for particular claims are an optional by-product rather than part of the semantics (as is standard when evaluating knowledge bases via abstract argumentation[CA07]).

The motivations behind the direct stable semantics are thus, in the first place, to provide a means of evaluating knowledge bases of strict and defeasible rules where, in particular, there may be conflicts between defeasible rules. Strict rules are interpreted as holding without exception, while conflicting defeasible rules give rise to different possible outcomes of the evaluation process. A further central motivation behind the direct stable semantics is that the semantics satisfies some basic desiderata for evaluating knowledge bases containing conflicting information, the so called rationality postulates [CA07], by construction (while adding further desiderata: groundedness and defeasible closure; see [SW17]).

Also, having arguments for particular claims as an optional by product, rather than an explicit component of the semantics, allows circumventing several shortcomings of evaluating knowledge bases via abstract argumentation. The main one being the potential exponential explosion of arguments that need to be constructed even for propositional knowledge bases in the more standard approaches. Related problems are the opacity of attacks and need of regeneration of the arguments when the knowledge base changes [SW17]. Finally, the direct stable semantics allows evaluating knowledge bases of rules allowing some limited first order features: first order predicates, variables, and constants (in this work we also add functions).

2.4.1 Syntax

As already indicated, the direct stable semantics is defined for defeasible theories or programs of strict and defeasible rules. The syntax of such theories is similar to that of logic programs as defined in Section 2.2.1. In particular, defeasible theories are also built from atoms and terms constructed on the basis of a language $\mathcal{L} = (\mathcal{V}, \mathcal{C}, \mathcal{F}, \mathcal{P})$ consisting of disjoint countable sets of variables $\mathcal{V}$, constant symbols $\mathcal{C}$, function symbols $\mathcal{F}$, and predicate symbols $\mathcal{P}$ as described in the latter section. Given the similarities between logical programs as defined in Section 2.2.1 and defeasible theories, when appropriate we will use terminology defined in the former section when referring to defeasible theories.

Rules of defeasible theories have the form

$$b_1, \ldots, b_n \rhd h$$

where h and each b_i $(1 \leq i \leq n)$ are positive literals of the form a or negative literals of the form $\neg a$ with a being an atom (as defined in Section 2.2.1). The connective "$\neg$" stands for strong negation rather than default or negation as failure. Also $\rhd \in \{\rightarrow, \Rightarrow\}$ with $\rightarrow$ standing for "strict implication" (similar to the *:-* connective for logical programs) and $\Rightarrow$ for "defeasible implication". As for rules of logical programs, $n \geq 0$; when $n = 0$ and $\rhd$ is $\rightarrow$ the rule is called a fact, while if $\rhd$ is $\Rightarrow$ the rule is an assumption. $B(r)$ denotes the set of (positive and negative) literals in the body of a rule r. S_π denotes the strict rules of π and D_π denotes the defeasible rules of a defeasible theory π.

Differences with logical programs are mainly hence the lack of disjunction in the heads of rules as well the use of strong negation instead of default negation. Crucially, defeasible theories allow the use of defeasible rules in addition to strict rules. The inteded interpretation of rules, similar as for logical programs, is as warranting inference of the atom in the head whenever each of the atoms in the body is justified. For negative literals of the form $\neg a$ this now means that $\neg a$ has been derived rather than a not having been derived. Whenever each of the atoms in the body of a rule is justified, this makes the rule applicable. Strict rules are to be applied without exception, while defeasible rules are to be applied in all non-exceptional (i.e. not leading to contradictions) circumstances.

2.4.2 Semantics

The semantics of defeasible theories, as for logical programs considered in Section 2.2.1, is defined in terms of the groundings of the theories. The grounding of a defeasible theory π being the set of rules σr obtained from applying, to each rule $r \in \pi$, all possible substitutions σ from the variables in r to elements of U_π; U_π being the set of all ground terms that can be constructed using constants and function symbols appearing in π.

We note that we slightly deviate from the definitions of defeasible theories and their semantics from [SW17] in that we allow function symbols and, hence, the possibility of infinite groundings of defeasible theories. Despite this crucial difference, we can make use of the original definitions from [SW17] without much further ado.

An interpretation for a defeasible theory π is a set $I \subseteq L_\pi$, L_π being all the literals that can be constructed from atoms appearing in π. Such an interpretation I satisfies a ground rule r iff $H(r) \in I$ whenever $B(r) \subseteq I$. Given a set of (strict and/or defeasible) ground rules R, the closure of an interpretation I under R on the other hand includes all the heads of rules which are applicable w.r.t. I, i.e.

$$R(I) := \{H(r) \mid B(r) \subseteq I \textit{ and } r \in R\}.$$

The interpretation I is closed under a set of rules R if $R(I) = I$.

The starting point of the definition of the direct stable semantics is the notion of "possible sets"; these single out sets of literals to which defeasible rules of a program of interest π are maximally applied while preserving consistency (this is the "defeasible closure" postulate we mentioned in the introduction to this section). Formally:

Definition 11. *Let π be a ground defeasible theory. A set of literals $M \subseteq L_\pi$ is a possible set for π iff there is a set $D_M \subseteq D_\pi$ such that:*

1. *M is consistent, i.e. for every atom a in M $a \in M$ iff $\neg a \notin M$;*
2. *M is closed under $S_\pi \cup D_M$;*
3. *D_M is maximal w.r.t. items 1-2, i.e. there is no D'_M s.t. $D_M \subset D'_M \subseteq D_\pi$ and $(S_\pi \cup D'_M)(M)$ is consistent.*

We will stick with using the notation D_M for the set of defeasible rules "induced by" a possible set M.

Stable sets are possible sets where the justification of all literals is grounded in facts and assumptions ("groundedness postulate" mentioned in the introduction to this section). This further constraint is in order to disallow unsupported literals or literals whose justification status is cyclical, which can occur for possible sets. The notion of the justification status of a literal being traceable back to facts and assumptions is cashed out in term of the notion of a derivation or argument for the literal. For a set of (strict or defeasible) rules R a derivation of a literal z w.r.t. R is a partial order $\leq$ on R and s.t. first of all $\leq$ has a greatest element r with $H(r) = z$. Also, for each rule $r \in R$ and each $b \in B(r)$ there is a $r' \in R$ s.t. $r' < r$ ($<$ is the strict partial order contained in $\leq$) and $H(r') = b$.

Definition 12. *Let π be a defeasible theory. A stable set for π is a possible set M s.t. for every $z \in M$ there is a derivation of z w.r.t. $S_\pi \cup D_M$.*

2.4.3 Reasoning and complexity

The main reasoning tasks for defeasible theories when evaluated under the direct-stable-semantics are analogous to the tasks defined for answer-set-programs (see Section 2.2.4). The existence and credulous reasoning problems for propositional defeasible theories (w.r.t direct stable sets) are Σ_2^P-complete, while skeptical reasoning is Π_2^P-complete [SW17]. With respect to a specific stable set and literal, computing an argument (or derivation) for the literal can be done in polynomial time. The complexity of first-order reasoning for defeasible theories w.r.t. the direct-stable-semantics remains to be investigated but the reduction we sketch in Section 4.2.4.2 gives a NEXPTIME upper-bound for theories without function symbols.

2.5 Computational complexity

Complexity theory studies computational problems from the perspective of the resources (running time, memory space) required to solve them. One of the main objectives has, in particular, been to classify computational problems from this perspective. We simply recall some basic notions of complexity theory we make use of in our work here and refer to e.g. [Pap07] and [AB09] for detailed expositions.

The traditional focus of complexity theory is on decision problems; these are problems that classify the input instances as either having a certain property (the "yes" instances) or not (the "no" instances). Also, the difficulty of problems is studied in terms of the worst-case behavior (in terms of running time, memory space) of algorithms that solve them. Worst case behaviour is measured as a function of the input-size. Also, in order to abstract from hardware specifics, algorithms are specified for Turing Machines, the prominent model of computation introduced by Turing in [Tur37] to formalise the notion of an "algorithm". An important distinction for complexity theory concerns deterministic and non-deterministic Turing machines. Though equally powerful from the perspective of the algorithms they can implement, these are relevant when considering the complexity of algorithms since to date no efficient (polynomial) simulation of non-determinism in a deterministic Turing machine has been shown to exist and it is widely assumed that it in fact does not.

The central theoretical tool in complexity theory is the notion of reduction. A reduction from a problem P_1 to a problem P_2 is a function f from the instances of problem P_1 to the instances of problem P_2 which is efficiently computable (for the purposes of this work: requires polynomial time) and which satisfies the following property: an instance i of P_1 has answer "yes" with respect to the problem P_1 if and only if the instance $f(i)$ of P_2 has answer "yes" with respect to the problem P_2.

The notion of reduction allows to order decision problems as follows: $P_1 \leq_R P_2$ if and only if there exists a reduction from P_1 to P_2. The intuition behind this ordering is that if there exists a reduction f from P_1 to P_2 then P_1 is not more difficult than P_2, since one can solve an instance i of P_1 by solving $f(i)$ of P_2. A problem P is called hard for a complexity class C if every problem in C can be reduced to P. It is complete for C (this can be written: C-c) if it also belongs to C. Problems that are complete for a certain class C can be considered "prototypical" for the class in the sense that any problem in C can be efficiently reduced to the problem in question. The complement of a class C is denoted co-C.

Some important complexity classes relevant for the present work are in the first place the class P of problems that can be solved by a deterministic Turing machine in polynomial time. The same definition for P but for non deterministic Turing machines leads to the class NP. As has been hinted at in a previous paragraph, it is generally assumed that P $\subset$ NP. While problems in class P are considered tractable ("efficiently solvable"), problems in NP are thus assumed to be intractable. The notion of NP-completeness stems from [Coo71] and [Lev73], where it is also shown that (in the case of Levin, a variant of) the satisfiability problem of propositional logic (SAT) is a prototypical problem for NP. A prototypical problem for coNP is the unsatisfiability problem for propositional logic, i.e. deciding for a propositional formula if it is unsatisfiable.

The class PSPACE is the set of problems that can be answered by a deterministic Turing machine using polynomial space. It is an open question but usually assumed that NP $\subset$ PSPACE. As already indicated in Section 2.1.4, in [SM73] it is shown that QSAT is PSPACE-complete. The polynomial hierarchy is a family of complexity classes within PSPACE introduced in [SM73, Sto76]. It is defined as follows for $k \geq 0$:

$$\Sigma_0^P = \Pi_0^P := \mathrm{P}$$

$$\Sigma_{k+1}^P := \mathrm{NP}^{\Sigma_k^P}, \Pi_{k+1}^P := \text{co-}\Sigma_{k+1}^P$$

Σ_{k+1}^P is the class of all problems that can be decided non-deterministically in polynomial time with the help of an oracle for a problem in Σ_k^P . An oracle is a subroutine which solves a problem in the complexity class Σ_k^P in constant time. In particular, one has that NP $= \Sigma_1^P$ and coNP $= \Pi_1^P$. The polynomial hierarchy PH is defined as the union $\cup_{k=0}^{\infty}\Sigma_k^P$. As also indicated in Section 2.1.4, for each level of the polynomial hierarchy there is a class of QBFs (identified by their prefix type) whose satisfiability problem is complete for that level of the polynomial hierarchy.

Other classes of problems referred to in this work are, first of all, the class DP of problems that are in the intersection of NP and coNP. A prototypical problem for this class is the problem of deciding for a pair of propositional formulas, whether one is satisfiable and the other unsatisfiable (the SAT-UNSAT problem). The class EXPTIME is the class of problems that can be decided in exponential time by a deterministic Turing machine, while NEXPTIME is the class of problems that can be decided in exponential time by a non-deterministic Turing machine. The class NEXPTIME^{NP} is then the class of problems that can be decided in NEXPTIME with an NP oracle. Finally, problems that are undecidable are those that cannot be decided by a Turing machine, i.e. for which no Turing machine which can decide every instance of the problem exists. The classical example of an undecidable problem is the satisfiability problem for first order logic [Tur37].

CHAPTER 3

Advanced QBF and ASP encodings for ADFs

In this chapter we present complexity and link information sensitive QBF encodings for ADFs in Section 3.1 before we turn to presenting dynamic ASP encodings for ADFs in Section 3.2.

3.1 Complexity and link information sensitive QBF encodings for ADFs

We start in this section by presenting encodings of reasoning tasks (verification, as well as credulous and skeptical acceptance problems; see Section 2.3) for ADFs into QBFs. This work extends that of [Dil14] in that we first present complexity sensitive encodings also for the stable semantics. This means that the complexity of the satisfiability problem for the QBFs that result from our encodings reflect the complexity of the several reasoning tasks (for the stable semantics) we encode. The complexity sensitive encodings for the stable semantics are presented in Section 3.1.2.

Secondly, in this section we present QBF encodings for the reasoning tasks for all main semantics for ADFs (admissible, complete, preferred, grounded, stable) that are "link information sensitive". This means that our encodings make use of additional information about the link types of the input ADF, whenever this information is available. The motivation behind the latter is that the additional information about the links may serve QSAT solvers being fed an implementation of our encodings to the same degree that the extra information may make the reasoning tasks easier. The latter is explained in detail in Section 3.1.3.1. The presentation of our novel link information sensitive encodings makes up the largest part of this section (Section 3.1.2).

For developing QBF encodings for the reasoning tasks we are interested in this work we mostly follow the same methodology. For each semantics $\sigma \in \{adm, com, prf, grd, stb\}$ we first construct a defining encoding function, usually denoted by $\mathcal{E}_\sigma$ or some primed version of $\mathcal{E}_\sigma$ (e.g. $\mathcal{E}'_\sigma$ or $\mathcal{E}'''_\sigma$). Given an ADF D, the defining encoding function for σ returns a QBF whose models correspond to the σ interpretations of D. Such defining encoding functions return the encoding of the enumeration problem for the different semantics and can be used in a generic fashion to provide encodings for the reasoning tasks (/decision problems) we consider in this work. We describe how to do so in Proposition 2 at the end of Section 3.1.1.1.

The alluded to method for providing encodings has the advantage of being general in the sense that the encodings of the reasoning tasks we provide via Proposition 2 for some semantics σ remain correct if the defining encoding function is redefined. In fact, the difference between many of the new link information sensitive encodings we present in Section 3.1.3 and the encodings we provided in [Dil14] is, precisely, that the encoding function is different. On the other hand, this method does not guarantee that we obtain encodings for the reasoning tasks that structurally reflect the complexity of the tasks.

In fact, for a few of the reasoning tasks we consider in this work the defining encoding functions we provide result in encodings that are not sensitive to the complexity of the tasks. These cases either are equivalent to some other reasoning task for which we provide encodings that are adequate with respect to the complexity, or otherwise we provide alternative encodings that more accurately reflect the complexity of the tasks.

Before moving on to the complexity sensitive QBF encodings for the stable semantics as well as our link information sensitive encodings we lay some common groundwork in Section 3.1.1. Specifically, we deal with some technical preliminaries in Section 3.1.1.1 and define some basic modules that we make repeated use of in our encodings in Section 3.1.1.2.

3.1.1 Groundwork

3.1.1.1 Preliminaries

As already indicated, we begin with some technical preliminaries. First of all, in order to avoid confusion, in this section of our work (Section 3.1) we distinguish between QBF and ADF interpretations by using different notations. Thus, $\hat{v}$ denotes a QBF interpretation, i.e. a two-valued interpretation over the set of propositional atoms $\mathcal{P}$ as introduced in Section 2.1. On the other hand, v (without the "hat") denotes a (two valued or three valued) interpretation over the set of statements of an ADF as explained in Section 2.3.

Turning to our encodings; in them we often need to introduce propositional variables to refer to different objects of some set T, e.g. statements of an ADF. For this we use *indexed* variables w.r.t. T. Given a set of propositional atoms $P \subset \mathcal{P}$, the set of indexed variables w.r.t. P and T is a mapping P_T from P to T. We denote the variable $p \in P$ mapped to $t \in T$ by p_t and thus will often (somewhat abusing the notation) treat the mapping P_T as a set; specifically, the set $P_T := \{p_t \mid t \in T\}$. Given a propositional formula (e.g. an acceptance condition of an ADF) ϕ with variables occurring in T, we then define ϕ^{P_T} as the propositional formula $\phi[t/p_t \mid t \in T]$, i.e. the formula that results from substituting each occurrence of t in ϕ by the corresponding $p_t \in P_T$.

To simplify the notation when referring to a set of indexed variables P_T we use the same letter-, in this case the letter "p",- both to refer to the set of indexed variables (here the "p" is in large case) as well as to refer to propositional atoms in P_T (the atoms are in small case, e.g. p_t for some $t \in T$). For instance A_T is the set $\{a_t \mid a \in A, t \in T\}$ (here the small case "a" is used to denote the atoms and the large case "A" to denote the set of indexed variables). We also sometimes use priming (also, multiple priming) or other superscripts rather than different letters to represent different sets of signed variables; e.g $P'_T := \{p'_t \mid p' \in P', t \in T\}$ and $P^{\oplus}{}_T := \{p^{\oplus}_t \mid p^{\oplus} \in P^{\oplus}, t \in T\}$. Except if stated otherwise, in this work we will assume that indexed variables with different representations are also disjoint. Thus, for instance, $P_T \cap Q_T = \emptyset$ and also $P'_T \cap P''_T = \emptyset$. On the other hand, we will sometimes use the representation P_R where $R \subseteq T$ to refer to the projection of the set of indexed variables P_T to R, i.e. $P_R := \{p_t \mid p \in P, t \in R\}$. From $R \subseteq T$ it clearly follows that also $P_R \subseteq P_T$.

To encode expressions about two valued interpretations on a set of statements S of an ADF within our QBF encodings, we will use indexed variables w.r.t. S. To encode expressions about *three* valued interpretations on S we follow the procedure also used in [AC13] and use signed indexed variables. These are two disjoint sets of *signed* variables $P^{\oplus}{}_T$ and $P^{\ominus}{}_T$ indexed w.r.t T; to simplify things we write the union of such disjoint sets of signed variables as $P^3_S := P^{\oplus}{}_T \cup P^{\ominus}{}_T = \{p_s{}^{\oplus} \mid p \in P^{\oplus}, s \in S\} \cup \{p_s{}^{\ominus} \mid p \in P^{\ominus}, s \in S\}$. The intended meaning of the indexed atom $p_s{}^{\oplus}$ is that the statement s is accepted, while the intended meaning of $p_s{}^{\ominus}$ that s is rejected under some interpretation for ADFs. The status of the statement s is undecided if both $p_s{}^{\oplus}$ and $p_s{}^{\ominus}$ are false.

ADF semantics are based on three or two truth values. Since there are four possible truth value assignments for a statement s via the variables $p_s{}^{\oplus}$ and $p_s{}^{\ominus}$ we use in our QBF encodings, we need to restrict attention to *coherent* interpretations for QBFs. These exclude the possibility for a model to satisfy both $p_s{}^{\oplus}$ and $p_s{}^{\ominus}$.

Definition 13. *Let P^3_S be a set of signed atoms indexed by a set (of ADF statements) S. A two valued interpretation $\hat{v}$ is* coherent *on P^3_S if there is no $s \in S$ such that $\hat{v}(p_s{}^{\oplus}) = \boldsymbol{t}$ and $\hat{v}(p_s{}^{\ominus}) = \boldsymbol{t}$.*

We now formally define the correspondence we require of the models of the QBFs returned by a defining encoding $\mathcal{E}_\sigma$ for a semantics σ and the σ interpretations of the ADFs.

Definition 14. *Let S be a set of statements and P_S a set of propositional atoms indexed by S. A two valued interpretation $\hat{v}$ on P_S corresponds to a* two valued *interpretation v on S, denoted as $\hat{v} \cong_{P_S} v$ if $v(s) = \hat{v}(p_s)$ for all $s \in S$. A coherent two valued interpretation $\hat{v}'$ on a set P^3_S of signed atoms indexed by S corresponds to a* three valued *interpretation v' on S, denoted as $\hat{v}' \cong_{P^3_S} v'$ if the following three conditions are met:*

- *$v'(s) = \boldsymbol{t}$ if and only if $\hat{v}'(p_s{}^{\oplus}) = \boldsymbol{t}$ and $\hat{v}'(p_s{}^{\ominus}) = \boldsymbol{f}$;*
- *$v'(s) = \boldsymbol{f}$ if and only if $\hat{v}'(p_s{}^{\oplus}) = \boldsymbol{f}$ and $\hat{v}'(p_s{}^{\ominus}) = \boldsymbol{t}$; and*
- *$v'(s) = \boldsymbol{u}$ if and only if $\hat{v}'(p_s{}^{\oplus}) = \boldsymbol{f}$ and $\hat{v}'(p_s{}^{\ominus}) = \boldsymbol{f}$.*

The following lemma is a straightforward consequence of Definition 14 we make often use of in the proofs of correctness of the encodings we provide in this section of our work.

Lemma 3. *Let S be a set of statements, $\hat{v}_1$ and $\hat{v}_2$ two valued (QBF) interpretations, and v_1 and v_2 three or two valued interpretations on S.*

1. *Assume that P^3_S and Q^3_S are disjoint, $\hat{v}_1$ is coherent on P^3_S, $\hat{v}_2$ is coherent on Q^3_S, $\hat{v}_1 \cong_{P^3_S} v_1$ and $\hat{v}_2 \cong_{Q^3_S} v_2$. Then $\hat{v} := \hat{v}_1[Q^3_S/\hat{v}_2(Q^3_S)]$ is also coherent on each of P^3_S and Q^3_S, and it is also the case that $\hat{v} \cong_{P^3_S} v_1$ and $\hat{v} \cong_{Q^3_S} v_2$.*
2. *Assume that $\hat{v}_1$ is coherent on P^3_S, $\hat{v}_1 \cong_{P^3_S} v_1$ and $\hat{v}_2 \cong_{P_S} v_2$. Then $\hat{v} := \hat{v}_1[P_S/\hat{v}_2(P_S)]$ is also coherent on P^3_S and it is also the case that $\hat{v} \cong_{P^3_S} v_1$ and $\hat{v} \cong_{P_S} v_2$.*
3. *Assume that $\hat{v}_2$ is coherent on P^3_S, $\hat{v}_1 \cong_{P_S} v_1$ and $\hat{v}_2 \cong_{P^3_S} v_2$. Then $\hat{v} := \hat{v}_1[P^3_S/\hat{v}_2(P^3_S)]$ is also coherent on P^3_S and it is also the case that $\hat{v} \cong_{P_S} v_1$ and $\hat{v} \cong_{P^3_S} v_2$.*
4. *Assume that P_S and Q_S are disjoint, $\hat{v}_1 \cong_{P_S} v_1$ and $\hat{v}_2 \cong_{Q_S} v_2$. Then for $\hat{v} := \hat{v}_1[Q_S/\hat{v}_2(Q_S)]$ it is also the case that $\hat{v} \cong_{P_S} v_1$ and $\hat{v} \cong_{Q_S} v_2$.*

Proof. (sketch) Let $T_1 = P^3_S$ and $T_2 = Q^3_S$ in item 1, $T_1 = P^3_S$ and $T_2 = P_S$ in item 2, $T_1 = P_S$ and $T_2 = P^3_S$ in item 3 and $T_1 = P_S$ and $T_2 = Q_S$ in item 4. Then all these items follow from the fact that $\hat{v}$ and $\hat{v}_1$ as defined in each of the items agree on the variables in T_1 while $\hat{v}$ and $\hat{v}_2$ agree on the variables in T_2 (and that T_1 and T_2 are disjoint). □

We are now in a position to provide the formal definition of a defining encoding function $\mathcal{E}_\sigma$ for a given semantics σ for ADFs. Given an ADF $D = (S, C)$, $\mathcal{E}_\sigma(D)$ returns a QBF whose every model corresponds (in the sense of Definition 14) to one of the σ interpretations of D ("soundness"). In turn, every σ interpretation corresponds to a model of ϕ ("completeness"). Given a set of propositional atoms P and depending on whether σ returns two valued or three valued interpretations, $\mathcal{E}_\sigma$ returns a QBF with free variables in a set of signed (P^3_S) or unsigned (P_S) variables standing for the statements of the ADFs respectively. We thus write $\mathcal{E}_\sigma[D, P^3_S, \ldots]$ or $\mathcal{E}_\sigma[D, P_S, \ldots]$ respectively for the application of the defining encoding function $\mathcal{E}_\sigma$ to the ADF D.

The dots in the notation $\mathcal{E}_\sigma[D, P^3_S, \ldots]$ and $\mathcal{E}_\sigma[D, P_S, \ldots]$ stand for the remaining variables on which the application of the defining encoding depends; for simplicity, we will often leave these unspecified. In addition to soundness and completeness, when σ returns three valued interpretations ($\sigma \in \{adm, com, prf, grd\}$), $\mathcal{E}_\sigma[D, P_S^3, \ldots]$ must also be a QBF whose models are coherent on P_S^3 in the sense of Definition 13.

Definition 15. *Let $\sigma \in \{adm, com, prf, grd, mod, stb\}$*[1]*. A* defining encoding function *for σ is a total function $\mathcal{E}_\sigma$ from ADFs to QBFs. Given an ADF $D = (S, C)$ and a set of atoms $P \subset \mathcal{P}$ it returns a QBF $\mathcal{E}_\sigma[D, Q, \ldots]$ with free variables $Q = P_S^3$ if $\sigma \in \{adm, com, prf, grd\}$ and $Q = P_S$ if $\sigma \in \{mod, stb\}$. Furthermore,*

1. *(Coherence) If $\sigma \in \{adm, com, prf, grd\}$, then any two valued interpretation $\hat{v}$ such that $\hat{v} \models \mathcal{E}_\sigma[D, P_S^3, \ldots]$ is coherent on P_S^3.*
2. *(Soundness)*
   a) *If $\sigma \in \{adm, com, prf, grd\}$ and $\hat{v}$ is a two valued interpretation such that $\hat{v} \models \mathcal{E}_\sigma[D, P_S^3, \ldots]$, then the three valued interpretation v on S such that $\hat{v} \cong_{P_S^3} v$ is a σ interpretation of D.*
   b) *If $\sigma \in \{mod, stb\}$ and $\hat{v}$ is a two valued interpretation s.t. $\hat{v} \models \mathcal{E}_\sigma[D, P_S, \ldots]$, then the two valued interpretation v on S such that $\hat{v} \cong_{P_S} v$ is a σ interpretation of D.*
3. *(Completeness)*
   a) *If $\sigma \in \{adm, com, prf, grd\}$ and v is a three valued σ interpretation of D, then for any two valued interpretation $\hat{v}$ such that $\hat{v} \cong_{P_S^3} v$, it holds that $\hat{v} \models \mathcal{E}_\sigma[D, P_S^3, \ldots]$.*
   b) *If $\sigma \in \{mod, stb\}$ and v is a two valued σ interpretation of D, then for any two valued interpretation $\hat{v}$ such that $\hat{v} \cong_{P_S} v$, it holds that $\hat{v} \models \mathcal{E}_\sigma[D, P_S, \ldots]$.*

Having a defining encoding function $\mathcal{E}_\sigma$ for a semantics σ, one automatically obtains encodings for all reasoning tasks for which we seek encodings in this work. This is captured in Proposition 2 (proof in [Dil14, DWW15]).

In the encoding of the verification problem in Proposition 2 we use the notation $\psi[T/v(S)]$ for a QBF ψ with free variables in $T = P_S^3$ or $T = P_S$. This notation stands for the replacement of these variables in ψ with the special logical constants $\top$ or $\bot$ in accordance to an ADF interpretation v on the statements S. Concretely, for $T = P_S^3$, each $p_s{}^\oplus$ is replaced with $\top$ if $v(s) = \mathbf{t}$, with $\bot$ if $v(s) = \mathbf{f}$ or $v(s) = \mathbf{u}$. In the same manner, each $p_s{}^\ominus$ is replaced with $\top$ if $v(s) = \mathbf{f}$, yet replaced with $\bot$ if $v(s) = \mathbf{t}$ or $v(s) = \mathbf{u}$. On the other hand, if $T = P_S$, then each p_s is replaced with $\top$ if $v(s) = \mathbf{t}$, with $\bot$ if $v(s) = \mathbf{f}$ or $v(s) = \mathbf{u}$.

[1]We also include the model semantics ("*mdl*") in this discussion although we do not develop QBF encodings for the model semantics in this work. The reason is that we do make use of a defining encoding function for the model semantics from [Dil14] in some of our QBF encodings for the other semantics.

Proposition 2. *Let $D = (S, C)$ be an ADF, $s^* \in S$, $\sigma \in \{adm, com, prf, grd, mod, stb\}$, and v a three (or two) valued interpretation. If $\sigma \in \{adm, com, prf, grd\}$, then let $x = {p_{s^*}}^{\oplus}$ and $T = P^3_S$. Otherwise if $\sigma \in \{mod, stb\}$, then let $x = p_{s^*}$ and $T = P_S$. Also, let $\mathcal{E}_\sigma$ be a defining encoding function for σ with free variables in T. It holds that*

- *$Cred_\sigma(D, s^*) = yes$ iff $\exists T(\mathcal{E}_\sigma[D, T, \ldots] \wedge x)$ is true;*
- *$Skept_\sigma(D, s^*) = yes$ iff $\forall T(\mathcal{E}_\sigma[D, T, \ldots] \rightarrow x)$ is true; and*
- *$Ver_\sigma(D, v) = yes$ iff $\mathcal{E}_\sigma[D, T, \ldots][T/v(S)]$ is true.*

3.1.1.2 Basic modules

When encoding the reasoning tasks associated to ADFs as QBFs we make repeated use of some simple modules. We start by defining some basic modules (most of which are from [Dil14]) that we make use of in this work.

In the first place, given a set of signed indexed variables P^3_S w.r.t. a set S of statements of an ADF, the following formula "filters out" QBF interpretations which are not coherent on P^3_S:

$$coh[P^3_S] := \bigwedge_{s \in S} \neg({p_s}^{\oplus} \wedge {p_s}^{\ominus}).$$

Formally, this is expressed in the following Lemma (proof in [Dil14, DWW15]):

Lemma 4. *Let P^3_S be a set of signed indexed variables for a set of statements S. A two valued interpretation $\hat{v}$ is coherent on P^3_S if and only if $\hat{v} \models coh[P^3_S]$.*

In order to encode the definitions of ADF semantics as QBFs we often need to express that $v(s) \leq_i v'(s)$ on all $s \in S$ for two interpretations of an ADF $D = (S, C)$. The formula

$$\leq_i [P^3_S, {P'}^3_S] := \bigwedge_{s \in S} (({p_s}^{\oplus} \rightarrow {p'_s}^{\oplus}) \wedge ({p_s}^{\ominus} \rightarrow {p'_s}^{\ominus}))$$

does precisely that assuming both v and v' are three valued and using the indexed signed variables P^3_S and ${P'}^3_S$ to implicitly refer to the mappings to truth values of v and v' respectively.

The formula

$$\leq_i [P^3_S, P'_S] := \bigwedge_{s \in S} (({p_s}^{\oplus} \rightarrow p'_s) \wedge ({p_s}^{\ominus} \rightarrow \neg p'_s))$$

encodes that $v \leq_i v'$ in case v is a three valued interpretation on a set of statements S and v' is a two valued interpretation on S. Finally,

$$\leq_i [P_S, {P'}^3_S] := \bigwedge_{s \in S} ((p_s \to {p'_s}^{\oplus}) \land (\neg p_s \to {p'_s}^{\ominus}))$$

encodes that $v \leq_i v'$ in case v is a two valued and v' a three valued interpretation on S.

Lemma 5 formally expresses the fact that $\leq_i [P^3_S, {P'}^3_S]$, $\leq_i [P^3_S, P_S]$, and $\leq_i [P_S, P^3_S]$ have the intended meanings. Straightforward proofs for $\leq_i [P^3_S, {P'}^3_S]$ as well as $\leq_i [P^3_S, P_S]$ (albeit, separated in proofs of distinct lemmas) can, again, be found in [Dil14, DWW15].

Lemma 5. *Let S be a set of statements, $T = P^3_S$ and $T' = {P'}^3_S$, $T = P^3_S$ and $T' = P'_S$, or $T = P_S$ and $T' = {P'}^3_S$ disjoint indexed variables.*

1. *Let $\hat{v}$ be a two valued interpretation such that $\hat{v} \models \leq_i [T, T']$. Moreover, $\hat{v}$ is also coherent on P^3_S if $T = P^3_S$ and coherent on ${P'}^3_S$ if $T' = {P'}^3_S$. Then for interpretations v, v' on S such that $\hat{v} \cong_T v$ and $\hat{v} \cong_{T'} v'$ it is the case that $v \leq_i v'$.*
2. *Let v and v' both be interpretations on S such that $v \leq_i v'$. Either both v and v' are three valued, or only one of v and v' is two-valued. If v is two-valued then $T = P_S$, otherwise $T = P^3_S$. If v' is two-valued then $T' = P'_S$, otherwise $T' = {P'}^3_S$. Then for any two valued interpretation $\hat{v}$ such that $\hat{v} \cong_T v$ and $\hat{v} \cong_{T'} v'$, it is the case that $\hat{v} \models \leq_i [T, T']$.*

Note that the reason Lemma 5 is split into two items instead of being written using an "if and only if" is that, given a set S of statements and indexed variables T and T' (with T and T' defined as in the lemma), there exist exactly two ADF interpretations v and v' on S that correspond to a (coherent) two valued interpretation $\hat{v}$ on T and T' respectively. On the other hand, for ADF interpretations v and v' defined as in the second item of Lemma 5, there exist (infinitely) many (coherent) two valued interpretations that correspond to v and v' (all variables that are not in T and T' can be assigned any truth value).

To conclude this section, we define "projected" versions of the modules $\leq_i [P^3_S, {P'}^3_S]$, $\leq_i [P^3_S, P_S]$, and $\leq_i [P_S, P^3_S]$. Given some set $Q \subseteq S$ e.g. $\leq_i [P^3_Q, {P'}^3_Q]$ is defined as follows:

$$\leq_i [P^3_Q, {P'}^3_Q] := \bigwedge_{s \in Q} (({p_s}^{\oplus} \to {p'_s}^{\oplus}) \land ({p_s}^{\ominus} \to {p'_s}^{\ominus}))$$

The definition of the projected versions of the modules $\leq_i [P^3_S, P_S]$ and $\leq_i [P_S, P^3_S]$ is defined in analogous manner.

3.1.2 Complexity sensitive encodings for the stable semantics

We turn now to the QBF encodings for the most important reasoning tasks w.r.t. ADFs. Since we left this function undefined in our previous work [Dil14], we start of by giving a complexity sensitive defining encoding function for the stable semantics. This function will also serve as a basis for the definition of our link information sensitive encodings for the stable semantics presented in Section 3.1.3.7.

Our defining encoding function for the stable semantics quite closely follows the definition of the stable semantics (Definition 9; albeit, via a syntactical trick) and, hence, requires defining encoding functions for the model and grounded semantics to be defined. Although the defining encoding function for the stable semantics we present works with any defining encoding functions for the model and grounded semantics, in order to obtain complexity sensitive encodings for the reasoning tasks, we use specific (complexity sensitive) defining encoding functions for both semantics from [Dil14].

The defining encoding function we make use of for the model semantics is the following:

$$\mathcal{E}_{mod}[D, P_S] := \bigwedge_{s \in S} (p_s \leftrightarrow {\phi_s}^{P_S}).$$

This defining encoding function pretty much recasts the definition of the model semantics (for an ADF $D = (S, \{\phi_s\}_{s \in S})$, $v \in mod(D)$ if $v(s) = v(\phi_s)$ for all $s \in S$) in QBF terms.

As to the defining encoding function for the grounded semantics, the version we use is based on the fact that (see [SW15]) $v \in grd(D)$ for an interpretation v and an ADF $D = (S, \{\phi_s\}_{s \in S})$ if v is the $\leq_i$-minimal interpretation satisfying i) for each $s \in S$ such that $v(s) = \mathbf{t}$ there exists an interpretation $w \in [v]_2$ for which $w(\phi_s) = \mathbf{t}$, ii) for each $s \in S$ such that $v(s) = \mathbf{f}$ there exists an interpretation $w \in [v]_2$ for which $w(\phi_s) = \mathbf{f}$, and finally iii) for each $s \in S$ such that $v(s) = \mathbf{u}$ there exist interpretations $w_1 \in [v]_2$ and $w_2 \in [v]_2$ such that $w_1(\phi_s) = \mathbf{t}$ and $w_2(\phi_s) = \mathbf{f}$. We call an interpretation v for an ADF D that satisfies properties i)-iii) a "candidate for being the grounded interpretation".

The encoding for the grounded semantics from [Dil14] that uses the characterisation of the grounded interpretation as the minimal candidate (w.r.t. $\leq_i$) for being the grounded interpretation is then as follows:

$$\begin{aligned} \mathcal{E}_{grd}[D, P^3_S, {P'}^3_S, P_S, P'_S] &:= coh[P^3_S] \wedge prop[D, P^3_S, P_S, P'_S] \wedge (\forall {P'}^3_S\ \omega) \\ \omega &:= \ ((coh[{P'}^3_S] \wedge prop[D, {P'}^3_S, P_S, P'_S]) \rightarrow \leq_i [P^3_S, {P'}^3_S]) \end{aligned}$$

The module $prop[.,.,.,.]$ encodes properties i)-iii) that candidates for being the grounded interpretation need to satisfy. For the ADF D and indexed variables P^3_S, P_S, and P'_S it is defined as follows:

$$
\begin{aligned}
prop[D, P^3_S, P_S, P'_S] &:= \bigwedge_{s \in S} (\psi_s \wedge \psi'_s \wedge \psi''_s) \\
\psi_s &:= ({p_s}^{\oplus} \rightarrow \exists P_S (\leq_i [P^3_S, P_S] \wedge {\phi_s}^{P_S})) \\
\psi'_s &:= ({p_s}^{\ominus} \rightarrow \exists P_S (\leq_i [P^3_S, P_S] \wedge \neg {\phi_s}^{P_S})) \\
\psi''_s &:= ((\neg {p_s}^{\oplus} \wedge \neg {p_s}^{\ominus}) \rightarrow \psi'''_s) \\
\psi'''_s &:= \exists P_S \cup P'_S (\leq_i [P^3_S, P_S] \wedge \leq_i [P^3_S, P'_S] \wedge {\phi_s}^{P_S} \wedge \neg {\phi_s}^{P'_S})
\end{aligned}
$$

Turning to our novel defining encoding function for the stable semantics; given an ADF $D = (S, \{\phi_s\}_{s \in S})$ this function is defined as detailed next:

$$
\begin{aligned}
\mathcal{E}_{stb}[D, P_S, {P'}^3_S, {P''}^3_S, P'_S, P''_S] &:= \ \psi \wedge \psi' \\
\psi &:= \mathcal{E}_{mod}[D, P_S] \wedge \bigwedge_{s \in S} (p_s \leftrightarrow {p'_s}^{\oplus}) \\
\psi' &:= \mathcal{E}_{grd}[(S, \{\phi_s \wedge p_s\}_{s \in S}), {P'}^3_S, {P''}^3_S, P'_S, P''_S]
\end{aligned}
$$

The encoding function maps the ADF D to a QBF consisting of three conjuncts. We first require that any model of the QBF correspond to a two valued (ADF-) model v of D. For this we use the defining encoding function for the model semantics $\mathcal{E}_{mod}[D, P_S]$. The second conjunct encodes that the statements mapped to $\mathbf{t}$ by v coincide with those mapped to $\mathbf{t}$ by the grounded interpretation of the reduct D^v as defined in Definition 9.

The last conjunct of $\mathcal{E}_{stb}[D, ., ., ., ., .]$ applies the defining encoding function $\mathcal{E}_{grd}$ of the grounded semantics to a modified version of D via a syntactic trick. The use of this syntactic trick is made explicit by modifying D in the first argument of $\mathcal{E}_{grd}[., ., ., ., .]$, giving us an (improper, as the acceptance conditions involve atoms which are not statements of the ADF) "ADF" $(S, \{\phi_s \wedge p_s\}_{s \in S})$. The meaning of this notation is that every indexed version of a modified acceptance condition $\phi_s \wedge p_s$ within the module $\mathcal{E}_{grd}[., ., ., ., .]$ leaves the atom p_s untouched; for instance $(\phi_s \wedge p_s)^{P'_S} := {\phi_s}^{P'_S} \wedge p_s$ and $(\phi_s \wedge p_s)^{P''_S} := {\phi_s}^{P''_S} \wedge p_s$.

Assume that we have a model of the QBF $\mathcal{E}_{stb}[D, P_S, {P'}^3_S, {P''}^3_S, P'_S, P''_S]$ which, because of the first conjunct of the QBF, corresponds to a two valued (ADF-) model v of D. The mentioned syntactic trick then essentially boils down to computing the grounded interpretation of the ADF $D^v_* := (S, C^* := \{\phi^*_s\}_{s \in S})$ with $\phi^*_s := \phi_s$ if $v(s) = \mathbf{t}$ and $\phi^*_s := \bot$ if $v(s) = \mathbf{f}$. We prove in Lemma 6 below that the grounded interpretation of D^v and D^v_* are equal when restricting attention to the assignments to the variables in $E_v = \{s \in S \mid v(s) = \mathbf{t}\}$.

Before proving Lemma 6 we first remind the reader that $\mathcal{U}^S$ is the interpretation on a set of statements S, distinguished by the fact that it maps each $s \in S$ to $\mathbf{u}$. Secondly, we introduce some auxiliary notation. For an ADF D and a two or three valued interpretation v, $E_v^D := \{s \mid s \in S \text{ and } v(s) = \mathbf{t}\}$. When it is clear from the context to which ADF D we are referring we will more often than not drop the D from the notation E_v^D, i.e. write E_v instead.

Also, let v and v' be two or three valued interpretations on $S \cup T_1$ and $S \cup T_2$ respectively for some sets S, T_1, and T_2. Then we define a form of projection on equality and inequalities: (i) $v =\downharpoonright_S v'$ if $v(s) = v'(s)$ for every $s \in S$, (ii) $v <_i\downharpoonright_S v'$ if $v(s) <_i v'(s)$ for every $s \in S$, and (iii) $v \leq_i\downharpoonright_S v'$ if $v(s) \leq_i v'(s)$ for every $s \in S$.

Lemma 6. *Let $D = (S, C = \{\phi_s\}_{s \in S})$ be an ADF and v a two valued interpretation on S. Then, $grd(D_*^v) =\downharpoonright_{E_v} grd(D^v)$ where $D_*^v := (S, C^* := \{\phi_s^*\}_{s \in S})$ with $\phi_s^* := \phi_s$ if $v(s) = \boldsymbol{t}$ and $\phi_s^* := \bot$ if $v(s) = \boldsymbol{f}$.*

Proof. We prove by induction on $n \geq 0$ that $\Gamma_{D_*^v}^n \leq_i\downharpoonright_{E_v} \Gamma_{D^v}^n$ and $\Gamma_{D_*^v}^{n+1} \geq_i\downharpoonright_{E_v} \Gamma_{D^v}^n$. From this then the lemma follows since consider a $k \geq 0$ such that $\Gamma_{D_*^v}^k =$ l.f.p. $(\Gamma_{D_*^v})$ and $\Gamma_{D^v}^k =$ l.f.p. (Γ_{D^v}). In that case, one has that $grd(D_*^v) =$ l.f.p. $(\Gamma_{D_*^v}) = \Gamma_{D_*^v}^k \leq_i\downharpoonright_{E_v} \Gamma_{D^v}^k =$ l.f.p. $(\Gamma_{D^v}) = grd(D^v)$ and $grd(D^v) =$ l.f.p. $(\Gamma_{D^v}) = \Gamma_{D^v}^k \leq_i\downharpoonright_{E_v} \Gamma_{D_*^v}^{k+1} = \Gamma_{D_*^v}^k =$ l.f.p. $(\Gamma_{D_*^v}) = grd(D_*^v)$. In other words, $grd(D_*^v) \leq_i\downharpoonright_{E_v} grd(D^v)$ and $grd(D_*^v) \geq_i\downharpoonright_{E_v} grd(D^v)$, hence $grd(D_*^v) =\downharpoonright_{E_v} grd(D^v)$.
The base case $(n = 0)$ of the inductive proof holds trivially since $\Gamma_{D_*^v}^0 = \mathcal{U}^S =\downharpoonright_{E_v} \mathcal{U}^{E_v} = \Gamma_{D^v}^0$ and $\Gamma_{D_*^v}^1 \geq_i\downharpoonright_{E_v} \Gamma_{D^v}^0 = \mathcal{U}^{E_v}$ since $\mathcal{U}^{E_v}$ is the $\leq_i$ least interpretation on E_V.
Before proving the inductive step note first that $\Gamma_{D_*^v}^1(s) = \mathbf{f}$ for every $s \in S$ such that $v(s) = \mathbf{f}$ (since for every $w \in [\mathcal{U}^S]_2$ it is the case that $w(\phi_s^*) = w(\bot) = \mathbf{f}$). Hence, since the characteristic operator is monotonic in fact $\Gamma_{D_*^v}^n(s) = \mathbf{f}$ and so also $w(s) = \mathbf{f}$ for every $w \in [\Gamma_{D_*^v}^n]_2$ holds for every $s \in S$ such that $v(s) = \mathbf{f}$ and each $n > 0$. A consequence of this is

- Observation A: $w(\phi_s) = w(\phi_s[b/\bot : v(b) = \mathbf{f}])$ for every $s \in E_v$, $w \in [\Gamma_{D_*^v}^n]_2$ and $n > 0$.

A second fact that will be of use in the proof of the inductive step is

- Observation B: if v and v' are two interpretations, $v \leq_i\downharpoonright_{S'} v'$ for some set S', and ψ is a propositional formula with all variables that occur in ψ being in S', then if for every $w \in [v]_2$ it is the case that $w(\psi) = x$ for $x \in \{\mathbf{t}, \mathbf{f}\}$ then also for every $w' \in [v']_2$ it holds that $w'(\psi) = x$.

This is because if there exists some $w' \in [v']_2$ such that $w'(\psi) \neq x$ $(x \in \{\mathbf{t}, \mathbf{f}\})$, then, since $v \leq_i\downharpoonright_{S'} v' \leq_i w'$ there also exists a $w \in [v]_2$ such that $w =\downharpoonright_{S'} w'$ and therefore $w(\psi) \neq x$ which is a contradiction.
Assume now (induction hypothesis) that $\Gamma_{D_*^v}^n \leq_i\downharpoonright_{E_v} \Gamma_{D^v}^n$ and $\Gamma_{D_*^v}^{n+1} \geq_i\downharpoonright_{E_v} \Gamma_{D^v}^n$ for $n > 0$. We first prove that then also $\Gamma_{D_*^v}^{n+1} \leq_i\downharpoonright_{E_v} \Gamma_{D^v}^{n+1}$. For this consider first that for an arbitrary

$s \in E_v$ it is the case that $\Gamma^{n+1}_{D^v_*}(s) = \mathbf{t}$. This means that for every $w \in [\Gamma^n_{D^v_*}]_2$ it holds that $w(\phi^*_s) = w(\phi_s) = \mathbf{t}$. Because of Observation A this also means that for every $w \in [\Gamma^n_{D^v_*}]_2$ it holds that $w(\phi_s[b/\bot : v(b) = \mathbf{f}]) = \mathbf{t}$. Now, since by induction hypothesis $\Gamma^n_{D^v_*} \leq_i\downharpoonright_{E_v} \Gamma^n_{D^v}$, by Observation B this implies that also for every $w' \in [\Gamma^n_{D^v}]_2$ it is the case that $w'(\phi_s[b/\bot : v(b) = \mathbf{f}]) = \mathbf{t}$ in which case $\Gamma^{n+1}_{D^v}(s) = \mathbf{t}$. In the same manner one can prove that if $\Gamma^{n+1}_{D^v_*}(s) = \mathbf{f}$ then also $\Gamma^{n+1}_{D^v}(s) = \mathbf{f}$ must be the case. Finally, if $\Gamma^{n+1}_{D^v_*}(s) = \mathbf{u}$ then $\Gamma^{n+1}_{D^v_*}(s) \leq_i \Gamma^{n+1}_{D^v}(s)$ whatever the value of $\Gamma^{n+1}_{D^v}(s)$. Since $s \in E_v$ was arbitrary one can conclude that $\Gamma^{n+1}_{D^v_*} \leq_i\downharpoonright_{E_v} \Gamma^{n+1}_{D^v}$.
To prove that $\Gamma^{n+2}_{D^v_*} \geq_i\downharpoonright_{E_v} \Gamma^{n+1}_{D^v}$ from the induction hypothesis assume now that for an arbitrary $s \in E_v$ it is the case that $\Gamma^{n+1}_{D^v}(s) = \mathbf{t}$. This means that for every $w \in [\Gamma^n_{D^v}]_2$ it holds that $w(\phi_s[b/\bot : v(b) = \mathbf{f}]) = \mathbf{t}$. By the induction hypothesis $\Gamma^{n+1}_{D^v_*} \geq_i\downharpoonright_{E_v} \Gamma^n_{D^v}$, hence by Observation B also for every $w' \in [\Gamma^{n+1}_{D^v_*}]_2$ it holds that $w'(\phi_s[b/\bot : v(b) = \mathbf{f}]) = \mathbf{t}$. Finally, by Observation A one has that in fact for every $w' \in [\Gamma^{n+1}_{D^v_*}]_2$ it is the case that $w'(\phi_s) = w'(\phi^*_s) = \mathbf{t}$. Hence, $\Gamma^{n+2}_{D^v_*}(s) = \mathbf{t}$. In the same manner one can prove that if $\Gamma^{n+1}_{D^v}(s) = \mathbf{f}$ also $\Gamma^{n+2}_{D^v_*}(s) = \mathbf{f}$. Finally, if $\Gamma^{n+1}_{D^v}(s) = \mathbf{u}$ then $\Gamma^{n+2}_{D^v_*}(s) \geq_i \Gamma^{n+1}_{D^v}(s)$ whatever the value of $\Gamma^{n+2}_{D^v_*}(s)$. Since $s \in E_v$ was arbitrary one can conclude that $\Gamma^{n+2}_{D^v_*} \geq_i\downharpoonright_{E_v} \Gamma^{n+1}_{D^v}$. □

We make use of Lemma 6 in the proof of Proposition 3, which states formally that $\mathcal{E}_{stb}[.,.,.,.,.,.]$ as defined above is indeed a defining encoding function for the stable semantics.

Proposition 3. *Given an ADF $D = (S, \{\phi_s\}_{s\in S})$ and the indexed variables P_S, P'^3_S, P''^3_S, P'_S, P''_S, let $\mathcal{E}_{stb}$ be the function returning the QBF with free variables in P_S, $\mathcal{E}_{stb}[D, P_S, P'^3_S, P''^3_S, P'_S, P''_S]$. Then $\mathcal{E}_{stb}$ is a defining encoding function for the stable semantics.*

Proof. Let $D = (S, C)$ be an ADF. In order to prove soundness consider a two valued interpretation $\hat{v}$ such that $\hat{v} \models \mathcal{E}_{stb}[D, P_S, P'^3_S, P''^3_S, P'_S, P''_S]$. Then $\hat{v} \models \mathcal{E}_{mod}[D, P_S]$ from which it follows via Lemma 3.2.4 in [Dil14] (Proposition 3.17 in [DWW15]) that $v \in mod(D)$ for the interpretation v such that $\hat{v} \cong_{P_S} v$. Now, it is also the case that $\hat{v} \models \psi'$ for $\psi' = \mathcal{E}_{grd}[(S, \{\phi_s \wedge p_s\}_{s\in S}), P'^3_S, P''^3_S, P'_S, P''_S]$ and hence by Lemma 2 $\hat{v} \models \psi'[P_S/\hat{v}(P_S)]$. Since $\hat{v}(p_s) = v(s)$ for all $s \in S$ the latter is equivalent to $\hat{v} \models \psi'[P_S/v(S)]$.

From Proposition 3.28 in [DWW15] (based on Lemma 3.6.3 from [Dil14]) it follows that $v' = grd(D^{\#})$ for the interpretation v' such that $\hat{v} \cong_{P'_S} v'$ and the ADF $D^{\#} = (S, \{\phi^{\#}_s\}_{s\in S})$ where $\phi^{\#}_s = \phi_s \wedge \top$ if $v(s) = \mathbf{t}$ and $\phi^{\#}_s = \phi_s \wedge \bot$ if $v(s) = \mathbf{f}$. Note that since $\phi_s \wedge \top \equiv \phi_s$ and $\phi_s \wedge \bot \equiv \bot$ for any $s \in S$ the ADF $D^{\#}$ is equivalent to the ADF $D^v_* := (S, C^* := \{\phi^*_s\}_{s\in S})$ defined as in Lemma 6. Hence, $v' = grd(D^v_*)$. Finally, consider an arbitrary $s \in S$ such that $v(s) = \mathbf{t}$, i.e. $s \in E_v$. Since $\hat{v} \models \bigwedge_{s\in S}(p_s \leftrightarrow {p'_s}^{\oplus})$, in particular $\hat{v} \models p_s \leftrightarrow {p'_s}^{\oplus}$. Hence, given that $\hat{v} \cong_{P_S} v$ and therefore $\hat{v}(p_s) = \mathbf{t}$, it must

also be the case that $\hat{v}({p'_s}^{\oplus}) = \mathbf{t}$. From this it follows that since $\hat{v} \cong_{P'_S} v'$ also $v'(s) = \mathbf{t}$ and since $s \in E_v$ was arbitrary in fact $E_v \subseteq E_{v'}$. In the same manner from the fact that $\hat{v} \models \bigwedge_{s\in S}(p_s \leftrightarrow {p'_s}^{\oplus})$ one can also conclude that $E_{v'} \subseteq E_v$ and, hence, $E_v = E_{v'}$. Since $v' = grd(D^v_*)$ from Lemma 6 it follows that $E^D_v = E^{D^v}_{v''}$ where $v'' = grd(D^v)$.
To prove completeness, let $D = (S, C)$ be an ADF and v a two valued interpretation such that $v \in stb(D)$. Then $v \in mod(D)$ and $E^D_v = E^{D^v}_{v''}$ for $v'' = grd(D^v)$. From Lemma 6 it follows that also $E^D_v = E^{D^v_*}_{v'}$ for $v' = grd(D^v_*)$. As has been argued for in the proof of soundness, the ADF D^v_* is equivalent to the ADF $D^{\#}$ defined as above and hence $v' = grd(D^{\#})$. Let $\hat{v}$ be a two valued interpretation such that $\hat{v} \cong_{P_S} v$ and $\hat{v} \cong_{P'_S} v'$. Then since $v \in mod(D)$ and $\hat{v} \cong_{P_S} v$ it follows from Lemma 3.2.4 in [Dil14] that $\hat{v} \models \mathcal{E}_{mod}[D, P_S]$. Also, since $v' = grd(D^{\#})$, one has, via Proposition 3.28 in [DWW15], that $\hat{v} \models \psi'[P_S/v(S)] = \psi'[P_S/\hat{v}(P_S)]$ for $\psi' = \mathcal{E}_{grd}[(S, \{\phi_s \wedge p_s\}_{s\in S}), {P'}^3_S, {P''}^3_S, P'_S, P''_S]$. This by Lemma 2 is equivalent to $\hat{v} \models \psi'$. Finally, from the fact that $E^D_v = E^{D^v_*}_{v'}$, $\hat{v} \cong_{P_S} v$, and $\hat{v} \cong_{P'_S} v'$ it is easy to show that $\hat{v}(p_s) = \mathbf{t}$ if and only if $\hat{v}({p'_s}^{\oplus}) = \mathbf{t}$ for each $s \in S$ and hence $\hat{v} \models \bigwedge_{s\in S}(p_s \leftrightarrow {p'_s}^{\oplus})$. In conclusion, $\hat{v}$ satisfies all of the conjuncts of $\mathcal{E}_{stb}[D, P_S, {P'}^3_S, {P''}^3_S, P'_S, P''_S]$ and, hence, $\hat{v} \models \mathcal{E}_{stb}[D, P_S, {P'}^3_S, {P''}^3_S, P'_S, P''_S]$. $\square$

Although in the definition of the defining encoding function $\mathcal{E}_{stb}$ we defined above we made use of the defining encoding functions $\mathcal{E}_{mod}$ and $\mathcal{E}_{grd}$, it is easy to adapt it to make use of any other versions of the defining encoding functions for the model and grounded semantics (note that in the proof of Proposition 3 we don't make use of any other property of $\mathcal{E}_{mod}$ and $\mathcal{E}_{grd}$ other than they are defining encoding functions). On the other hand, when making use of the specific defining encoding functions $\mathcal{E}_{mod}$ and $\mathcal{E}_{grd}$, the encoding $\mathcal{E}_{stb}[D, P_S, {P'}^3_S, {P''}^3_S, P'_S, P''_S]$ can be simplified as we spell out in detail now.

Observe first of all that when we expand $\mathcal{E}_{grd}[(S, \{\phi_s \wedge p_s\}_{s\in S}), {P'}^3_S, {P''}^3_S, P'_S, P''_S]$ within the definition of $\mathcal{E}_{stb}[D, P_S, {P'}^3_S, {P''}^3_S, P'_S, P''_S]$ above we obtain

$$\begin{aligned}
&\mathcal{E}_{stb}[D, P_S, {P'}^3_S, {P''}^3_S, P'_S, P''_S] = \ \psi \wedge \psi'_1 \wedge \psi'_2\\
&\psi = \mathcal{E}_{mod}[D, P_S] \wedge \bigwedge_{s\in S}(p_s \leftrightarrow {p'_s}^{\oplus})\\
&\psi'_1 = coh[{P'}^3_S] \wedge prop[(S, \{\phi_s \wedge p_s\}_{s\in S}), {P'}^3_S, P'_S, P''_S]\\
&\psi'_2 = \forall {P''}^3_S((coh[{P''}^3_S] \wedge prop[(S, \{\phi_s \wedge p_s\}_{s\in S}), {P''}^3_S, P'_S, P''_S]) \rightarrow_{\leq_i} [{P'}^3_S, {P''}^3_S])
\end{aligned}$$

Now, we note that any $v \in mod(D)$ is a candidate to being the grounded interpretation of the ADF $D^v_* = (S, C^* := \{\phi^*_s\}_{s\in S})$ defined as in Lemma 6. In effect, if s is a statement of D^v_* s.t. $v(s) = \mathbf{t}$ then there is an interpretation $w = v \in [v]_2$ s.t. $w(\phi^*_s) = w(\phi_s) = \mathbf{t}$ (since $v \in mod(D)$, $v(s) = v(\phi_s)$). On the other hand, if s is a statement of D^v_* s.t. $v(s) = \mathbf{f}$ then there is an interpretation $w = v \in [v]_2$ s.t. $w(\phi^*_s) = w(\bot) = \mathbf{f}$ ($w(\bot) = \mathbf{f}$ by the semantics of $\bot$). Also, for no statement s of D^v_* $v(s) = \mathbf{u}$, since v is a two-valued interpretation.

We have already argued that given an ADF model v corresponding to a QBF interpretation satisfying $\mathcal{E}_{mod}[D, P_S]$, applying the encoding for the grounded semantics $\mathcal{E}_{grd}$ on the modified ADF $(S, \{\phi_s \wedge p_s\}_{s\in S})$ essentially boils down to computing the grounded interpretation of D^v_* (see the proof of Proposition 3). Together with the fact that any $v \in mod(D)$ is a candidate to being the grounded interpretation of the ADF $D^v_* = (S, C^* := \{\phi^*_s\}_{s\in S})$, we have that the conjuncts $\bigwedge_{s\in S}(p_s \leftrightarrow {p'}_s^{\oplus}) \wedge \psi'_1$ above are redundant, i.e.

$$\mathcal{E}_{mod}[D, P_S] \wedge \bigwedge_{s\in S}(p_s \leftrightarrow {p'}_s^{\oplus}) \wedge \psi'_1 \equiv \mathcal{E}_{mod}[D, P_S]$$

for

$$\psi'_1 = \ coh[{P'}^3_S] \wedge prop[(S, \{\phi_s \wedge p_s\}_{s\in S}), {P'}^3_S, {P'}_S, {P''}_S].$$

We need to use fewer sets of indexed variables and use the module $\leq_i [P_S, {P'}^3_S]$ rather than $\leq_i [{P'}^3_S, {P''}^3_S]$ in

$$\psi'_2 = \ \forall {P''}^3_S((coh[{P''}^3_S] \wedge prop[(S, \{\phi_s \wedge p_s\}_{s\in S}), {P''}^3_S, {P'}_S, {P''}_S]) \rightarrow \leq_i [{P'}^3_S, {P''}^3_S])$$

to obtain a simplified defining encoding function

$$\begin{aligned}\mathcal{E}'_{stb}[D, P_S, {P'}^3_S, {P'}_S, {P''}_S] &:= \ \mathcal{E}_{mod}[D, P_S] \wedge \psi \\ \psi := \forall {P'}^3_S((coh[{P'}^3_S] \wedge prop[(S, \{\phi_s \wedge p_s\}_{s\in S}), &{P'}^3_S, {P'}_S, {P''}_S]) \rightarrow \leq_i [P_S, {P'}^3_S]).\end{aligned}$$

As a final simplification, note that we can streamline the module $prop[.,.,.,.]$ within our encoding by replacing the modules $\leq_i [.,.]$ within $prop[.,.,.,.]$ with projected versions as explained in Section 3.1.1.2:

$$\begin{aligned}prop'[D, P^3_S, P_S, {P'}_S] &:= \bigwedge_{s\in S}(\psi_s \wedge \psi'_s \wedge \psi''_s) \\ \psi_s &:= ({p_s}^{\oplus} \rightarrow \exists P_{par_D(s)}(\leq_i [P^3_{par_D(s)}, P_{par_D(s)}] \wedge {\phi_s}^{P_{par_D(s)}})) \\ \psi'_s &:= ({p_s}^{\ominus} \rightarrow \exists P_{par_D(s)}(\leq_i [P^3_{par_D(s)}, P_{par_D(s)}] \wedge \neg{\phi_s}^{P_{par_D(s)}}))\end{aligned}$$

$$\begin{aligned}
\psi''_s &:= ((\neg {p_s}^{\oplus} \land \neg {p_s}^{\ominus}) \to \psi'''_s)\\
\psi'''_s &:= \exists P_{par_D(s)} \cup P'_{par_D(s)}(\psi''''_s \land \psi'''''_s)\\
\psi''''_s &:= \leq_i [P^3_{par_D(s)}, P_{par_D(s)}] \land {\phi_s}^{P_{par_D(s)}}\\
\psi'''''_s &:= \leq_i [P^3_{par_D(s)}, P'_{par_D(s)}] \land \neg{\phi_s}^{P'_{par_D(s)}})
\end{aligned}$$

We thus obtain the final form of our abridged version of the encoding for the stable semantics (here we simply replaced *prop*$[, , ., ., .]$ with *prop*$'[, , ., ., .]$).

$$\begin{aligned}
\mathcal{E}''_{stb}[D, P_S, P'^3_S, P'_S, P''_S] :=\ & \mathcal{E}_{mod}[D, P_S] \land \psi\\
\psi := \forall P'^3_S((coh[P'^3_S] \land prop'[(S, \{\phi_s \land p_s\}_{s\in S}), P'^3_S, P'_S, P''_S]) &\to \leq_i [P_S, P'^3_S]).
\end{aligned}$$

It should be relatively easy to see that this function is equivalent to the initial defining encoding function $\mathcal{E}_{stb}$.

Example 3. *In the following we spell out the QBF that results from applying the defining encoding function $\mathcal{E}''_{stb}$ to the ADF $D = (S, \{\phi_s\}_{s\in S})$ from Example 1. In the example we use the following sets of indexed variables:*

$$\begin{aligned}
P_S =& \{p_a, p_b, p_c\}\\
P'^3_S =& \{{p'_a}^{\oplus}, {p'_a}^{\ominus}, {p'_b}^{\oplus}, {p'_b}^{\ominus}, {p'_c}^{\oplus}, {p'_c}^{\ominus}\}\\
P'_S =& \{p'_a, p'_b, p'_c\}\\
P''_S =& \{p''_a, p''_b, p''_c\}
\end{aligned}$$

The encoding is:

$$\begin{aligned}
\mathcal{E}'_{stb}[D, P_S, P'^3_S, P'_S, P''_S] =\ & \mathcal{E}_{mod}[D, P_S] \land \psi\\
\psi =\ & \forall P'^3_S((coh[P'^3_S] \land prop[(S, \{\phi_s \land p_s\}_{s\in S}), P'^3_S, P'_S, P''_S]) \to \leq_i [P_S, P'^3_S])\\
\mathcal{E}_{mod}[D, P_S] =\ & ((p_a \leftrightarrow (p_b \lor \neg p_b)) \land (p_b \leftrightarrow p_b) \land (p_c \leftrightarrow (p_c \to p_b)))\\
coh[P'^3_S] =\ & \neg({p'_a}^{\oplus} \land {p'_a}^{\ominus}) \land \neg({p'_b}^{\oplus} \land {p'_b}^{\ominus}) \land \neg({p'_c}^{\oplus} \land {p'_c}^{\ominus})\\
prop'[(S,\{\phi_s \land p_s\}_{s\in S}), P'^3_S, P'_S, P''_S]) =\ & \Psi_a \land \Psi_b \land \Psi_c\\
\Psi_a =\ & (\psi_a \land \psi'_a \land \psi''_a)\\
\psi_a =\ & ({p'_a}^{\oplus} \to \exists\{p'_b\}((({p'_b}^{\oplus} \to p'_b) \land ({p'_b}^{\ominus} \to \neg p'_b)) \land ((p'_b \lor \neg p'_b) \land p_a)))
\end{aligned}$$

$$
\begin{aligned}
\psi'_a &= ({p'}_a^{\ominus} \rightarrow \exists\{{p'}_b\}((({p'}_b^{\oplus} \rightarrow {p'}_b) \wedge ({p'}_b^{\ominus} \rightarrow \neg {p'}_b)) \wedge \neg(({p'}_b \vee \neg {p'}_b) \wedge p_a)))\\
\psi''_a &= ((\neg {p'}_a^{\oplus} \wedge \neg {p'}_a^{\ominus}) \rightarrow \psi'''_a)\\
\psi'''_a &= \exists\{{p'}_b, {p''}_b\}(\psi''''_a \wedge \psi'''''_a)\\
\psi''''_a &= ((({p'}_b^{\oplus} \rightarrow {p'}_b) \wedge ({p'}_b^{\ominus} \rightarrow \neg {p'}_b)) \wedge (({p'}_b \vee \neg {p'}_b) \wedge p_a))\\
\psi'''''_a &= ((({p'}_b^{\oplus} \rightarrow {p''}_b) \wedge ({p'}_b^{\ominus} \rightarrow \neg {p''}_b)) \wedge \neg(({p''}_b \vee \neg {p''}_b) \wedge p_a))\\
\Psi_b &= (\psi_b \wedge \psi'_b \wedge \psi''_b)\\
\psi_b &= ({p'}_b^{\oplus} \rightarrow \exists\{{p'}_b\}((({p'}_b^{\oplus} \rightarrow {p'}_b) \wedge ({p'}_b^{\ominus} \rightarrow \neg {p'}_b)) \wedge ({p'}_b \wedge p_b)))\\
\psi'_b &= ({p'}_b^{\ominus} \rightarrow \exists\{{p'}_b\}((({p'}_b^{\oplus} \rightarrow {p'}_b) \wedge ({p'}_b^{\ominus} \rightarrow \neg {p'}_b)) \wedge \neg({p'}_b \wedge p_b)))\\
\psi''_b &= ((\neg {p'}_b^{\oplus} \wedge \neg {p'}_b^{\ominus}) \rightarrow \psi'''_b)\\
\psi'''_b &= \exists\{{p'}_b, {p''}_b\}(\psi''''_b \wedge \psi'''''_b)\\
\psi''''_b &= ((({p'}_b^{\oplus} \rightarrow {p'}_b) \wedge ({p'}_b^{\ominus} \rightarrow \neg {p'}_b)) \wedge ({p'}_b \wedge p_b))\\
\psi'''''_b &= ((({p'}_b^{\oplus} \rightarrow {p''}_b) \wedge ({p'}_b^{\ominus} \rightarrow \neg {p''}_b)) \wedge \neg({p''}_b \wedge p_b))\\
\Psi_c &= (\psi_c \wedge \psi'_c \wedge \psi''_c)\\
\psi_c &= ({p'}_c^{\oplus} \rightarrow \exists\{{p'}_b, {p'}_c\}((\psi_{c,b} \wedge \psi_{c,c} \wedge (({p'}_c \rightarrow {p'}_b) \wedge p_c))))\\
\psi_{c,b} &= ({p'}_b^{\oplus} \rightarrow {p'}_b) \wedge ({p'}_b^{\ominus} \rightarrow \neg {p'}_b))\\
\psi_{c,c} &= ({p'}_c^{\oplus} \rightarrow {p'}_c) \wedge ({p'}_c^{\ominus} \rightarrow \neg {p'}_c))\\
\psi'_c &= ({p'}_c^{\ominus} \rightarrow \exists\{{p'}_b, {p'}_c\}((\psi_{c,b} \wedge \psi_{c,c} \wedge \neg(({p'}_c \rightarrow {p'}_b) \wedge p_c))))\\
\psi''_c &= ((\neg {p'}_c^{\oplus} \wedge \neg {p'}_c^{\ominus}) \rightarrow \psi'''_c)\\
\psi'''_c &= \exists\{{p'}_b, {p''}_b, {p'}_c, {p''}_c\}(\psi''''_c \wedge \psi'''''_c)\\
\psi''''_c &= ((\psi_{c,b} \wedge \psi_{c,c} \wedge (({p'}_c \rightarrow {p'}_b) \wedge p_c)))\\
\psi'''''_c &= ((\psi'_{c,b} \wedge \psi'_{c,c} \wedge \neg(({p''}_c \rightarrow {p''}_b) \wedge p_c)))\\
\psi'_{c,b} &= ({p'}_b^{\oplus} \rightarrow {p''}_b) \wedge ({p'}_b^{\ominus} \rightarrow \neg {p''}_b))\\
\psi'_{c,c} &= ({p'}_c^{\oplus} \rightarrow {p''}_c) \wedge ({p'}_c^{\ominus} \rightarrow \neg {p''}_c))\\
\leq_i [P_S, {P'}^3_S] &= \tau_a \wedge \tau_b \wedge \tau_c\\
\tau_a &= ((p_a \rightarrow {p'}_a^{\oplus}) \wedge (\neg p_a \rightarrow {p'}_a^{\ominus}))\\
\tau_b &= ((p_b \rightarrow {p'}_b^{\oplus}) \wedge (\neg p_b \rightarrow {p'}_b^{\ominus}))\\
\tau_c &= ((p_c \rightarrow {p'}_c^{\oplus}) \wedge (\neg p_c \rightarrow {p'}_c^{\ominus}))
\end{aligned}
$$

We turn now to the complexity sensitive encodings for the reasoning tasks that are of interest to this work for the stable semantics. Just as for the remaining encodings of reasoning tasks we give in this work, we will give the encodings in prenex normal form (PNF); the main reason being that the complexity of the encodings can then be read of the prefix of the encodings (see Section 2.1.4). Also, most of todays QSAT solvers expect the input QBFs to be in a variant of this format.

Most prenex normal form QBF encodings of reasoning tasks we give in this work are obtained from a defining encoding function via Proposition 2 and the equivalences from Proposition 1. As is to be expected, the encodings for the stable semantics we give now are based on the defining encoding function $\mathcal{E}''_{stb}$.

To encode the reasoning tasks relevant to this work w.r.t the stable semantics in PNF, we first of all need the quantifier free or matrix form of the module $prop[.,.,.,.]$ (actually, we give the matrix form of the simplified version $prop'[.,.,.,.]$). For the definition we assume that the statements of the ADF $D = (S, \{\phi_s\}_{s \in S})$ of interest are numbered, i.e. the statements are $S = \{s_1, \dots, s_n\}$ $(n > 0)$.

To simplify our rendering of QBF encodings throughout this work, we here also introduce a notation that makes "lists" of sets of variables out of "blocks" of sets of variables. I.e. given a block of variables $V = V_1 \dots V_m$, $V^{\mathcal{L}} := V_1, \dots, V_m$.

$$\begin{aligned}
prop^M[D, P^3_S, {V_1}^{\mathcal{L}}, \dots, {V_n}^{\mathcal{L}}] :=& \bigwedge_{1 \leq i \leq n} (\psi_{s_i} \wedge \psi'_{s_i} \wedge \psi''_{s_i}) \\
V_i :=&\ {P^{s'_i}}_{par_D(s_i)}\ {P^{s''_i}}_{par_D(s_i)}\ {P^{s'''_i}}_{par_D(s_i)}\ {P^{s''''_i}}_{par_D(s_i)} \quad (1 \leq i \leq n) \\
\psi_{s_i} :=&\ ({p_{s_i}}^{\oplus} \rightarrow (\leq_i [P^3_{par_D(s_i)}, {P^{s'_i}}_{par_D(s_i)}] \wedge {\phi_s}^{{P^{s'_i}}_{par_D(s_i)}})) \quad (1 \leq i \leq n) \\
\psi'_{s_i} :=&\ ({p_{s_i}}^{\ominus} \rightarrow (\leq_i [P^3_{par_D(s_i)}, {P^{s''_i}}_{par_D(s_i)}] \wedge \neg {\phi_{s_i}}^{{P^{s''_i}}_{par_D(s_i)}})) \quad (1 \leq i \leq n) \\
\psi''_{s_i} :=&\ ((\neg {p_{s_i}}^{\oplus} \wedge \neg {p_{s_i}}^{\ominus}) \rightarrow \psi'''_{s_i}) \quad (1 \leq i \leq n) \\
\psi'''_{s_i} :=&\ \psi''''_{s_i} \wedge \psi'''''_{s_i} \quad (1 \leq i \leq n) \\
\psi''''_{s_i} :=&\ (\leq_i [P^3_{par_D(s_i)}, {P^{s'''_i}}_{par_D(s_i)}] \wedge \leq_i [P^3_{par_D(s_i)}, {P^{s''''_i}}_{par_D(s_i)}] \quad (1 \leq i \leq n) \\
\psi'''''_{s_i} :=&\ {\phi_{s_i}}^{{P^{s'''_i}}_{par_D(s_i)}} \wedge \neg {\phi_{s_i}}^{{P^{s''''_i}}_{par_D(s_i)}}) \quad (1 \leq i \leq n)
\end{aligned}$$

For an ADF $D = (S, \{\phi_s\}_{s \in S})$ and an interpretation v, the verification problem w.r.t. the stable semantics can now be encoded as follows:

$$\begin{aligned}
&Ver_{stb}(D, v) \equiv \forall V(\mathcal{E}_{mod}[D, P_S] \wedge ((coh[{P'}^3_S] \wedge \psi) \rightarrow \leq_i [P_S, {P'}^3_S]))[P_S/v(S)] \\
&\quad V := {P'}^3_S\ V_1 \dots\ V_n \\
&\quad V_i := {P^{s'_i}}_{par_D(s_i)}\ {P^{s''_i}}_{par_D(s_i)}\ {P^{s'''_i}}_{par_D(s_i)}\ {P^{s''''_i}}_{par_D(s_i)} \quad (1 \leq i \leq n) \\
&\quad \psi := prop^M[(S, \{\phi_s \wedge p_s\}_{s \in S}), {P'}^3_S, {V_1}^{\mathcal{L}}, \dots, {V_n}^{\mathcal{L}}].
\end{aligned}$$

The encoding returns a Π_1 QBF and thus matches the complexity of the verification problem w.r.t. the stable semantics which is coNP-complete.

As to the acceptance problems, credulous reasoning w.r.t the stable semantics and for a statement $s^* \in S$ can be encoded as

$$
\begin{aligned}
&Cred_{stb}(D, s^*) \equiv\ \exists P_S \forall V(\mathcal{E}_{mod}[D, P_S] \wedge ((coh[{P'}^3_S] \wedge \psi) \rightarrow_{\leq_i} [P_S, {P'}^3_S]) \wedge p_{s^*})\\
&\quad V := {P'}^3_S\ V_1\ \ldots\ V_n\\
&\quad V_i := {P^{s'_i}}_{par_D(s_i)}\ {P^{s''_i}}_{par_D(s_i)}\ {P^{s'''_i}}_{par_D(s_i)}\ {P^{s''''_i}}_{par_D(s_i)}\ \ (1 \leq i \leq n)\\
&\quad \psi := prop^M[(S, \{\phi_s \wedge p_s\}_{s \in S}), {P'}^3_S, {V_1}^{\mathcal{L}}, \ldots, {V_n}^{\mathcal{L}}].
\end{aligned}
$$

The encoding for credulous reasoning returns a Σ_2 QBF again matching the complexity of this decision problem w.r.t the stable semantics (Σ_2^P-complete). Finally, also the following Π_2 QBF encoding for skeptical reasoning w.r.t the stable semantics matches the complexity of this decision problem (Π_2^P-complete):

$$
\begin{aligned}
&Skept_{stb}(D, s^*) \equiv\ \forall P_S \exists V((\mathcal{E}_{mod}[D, P_S] \wedge ((coh[{P'}^3_S] \wedge \psi) \rightarrow_{\leq_i} [P^3_S, {P'}^3_S])) \rightarrow p_{s^*})\\
&\quad V_i := {P'}^3_S\ V_1\ \ldots\ V_n\\
&\quad V_i := {P^{s'_i}}_{par_D(s_i)}\ {P^{s''_i}}_{par_D(s_i)}\ {P^{s'''_i}}_{par_D(s_i)}\ {P^{s''''_i}}_{par_D(s_i)}\ \ (1 \leq i \leq n)\\
&\quad \psi := prop^M[(S, \{\phi_s \wedge p_s\}_{s \in S}), {P'}^3_S, {V_1}^{\mathcal{L}}, \ldots, {V_n}^{\mathcal{L}}].
\end{aligned}
$$

3.1.3 Link information sensitive encodings for ADFs

3.1.3.1 Link types and subclasses of ADFs

Links in ADFs can be classified as being either attacking, supporting, redundant or dependent. The importance of this classification of links is, first of all, (roughly) their correspondence with crucial notions in argumentation where e.g. one argument can be said to be attacking another. Alternatively, a group of arguments can jointly support another argument; the relation of one argument supporting another thus being dependent on the acceptance status of further arguments[2].

The classification of links is also important because the complexity of most reasoning tasks for ADFs can be usefully parametrised by the number of links which are neither attacking nor supporting (these are the dependent links) as well as the number of links

[2]See [BES+18] for further examples on the kinds of relationships between arguments (or, more accurately, statements) that can be represented using ADFs.

whose type is unknown. In particular, the complexity of most reasoning tasks for ADFs drops one level of the polynomial hierarchy when the number of such links occurring in an ADF is bounded by a fixed constant.

Subclasses of ADFs result when restrictions are put on the types of links that can appear in an ADF or the number of links which are of a certain type. Two such subclasses are Dung's AFs, where all links are attacking, and bipolar ADFs or BADFs where all links are required to be either attacking or supporting. Dung's AFs are still the most popular abstract argumentation formalism on which most other argumentation formalisms, including ADFs, are based. The interest of BADFs on the other hand is that, while being more general than AFs[3], they nevertheless enjoy the same computational complexity as AFs for many semantics. Moreover, several existing generalisations of AFs can be translated to BADFs [Pol16].

BADFs can be generalised by allowing up to k links which are neither attacking nor supporting for some fixed constant $k \geq 0$; such ADFs have been called k-BADFs [LMN$^+$18a]. Adding an epistemic dimension, a further generalisation is to allow up to k links which are neither attacking nor supporting or, alternatively, whose type is unknown. For simplicity, we call such links "hard links". As hinted at previously, the complexity of reasoning also for ADFs with a number of hard links that is bound by a fixed constant roughly drops one level of the polynomial hierarchy with respect to general ADFs.

Definition 16 makes the classification of links in ADFs according to whether they are supporting, attacking, redundant, or dependent precise. The intuition behind a link from a statement s to a statement t being supporting is that accepting s doesn't make t unaccepted if t was already accepted, given the acceptance status of the (other) parents of t. Similarly, a link from a statement s to a statement t being attacking is that accepting s doesn't make t accepted if t was unaccepted before.

Links which are both supporting and attacking are called redundant, the reason being that the acceptance status of a statement t that is connected to a statement s via a redundant link is independent of the acceptance status of s (i.e. the acceptance status of t does not change if the acceptance status of s changes). Dependent links are those which are neither supporting nor attacking; intuitively this means that parents of a statement t on such links are attacking or supporting depending on the acceptance status of other parents of t.

Definition 16. *Let $D = (S, \{\phi_s\}_{s \in S})$ be an ADF.*

[3]This is especially clear from the perspective of realisability which is the study of the sets of interpretations that can be realised by a given formalism under a semantics. A set of interpretations V can be realised by a formalism $\mathcal{F}$ under a semantics σ whenever there is some element $F \in \mathcal{F}$ -,an argumentation framework in the case of argumentation,- such that $\sigma(F) = V$. BADFs are strictly "in between" AFs and ADFs from the perspective of realisability, with some interpretation sets being realisable by BADFs but not by AFs under the main semantics, while there are also interpretation sets which are realisable by ADFs and not by BADFs[Str15a, Str15b, LPS16].

- *A link $(s,t) \in L_D$ is supporting (alternatively, s is a supporter of t) if for every (two valued) interpretation v, if $v(\phi_t) = \boldsymbol{t}$, then also $v[s/\boldsymbol{t}](\phi_t) = \boldsymbol{t}$.*
- *A link $(s,t) \in L_D$ is attacking (alternatively, s is an attacker of t) if for every (two valued) interpretation v, if $v(\phi_t) = \boldsymbol{f}$, then also $v[s/\boldsymbol{t}](\phi_t) = \boldsymbol{f}$.*
- *A link $(s,t) \in L_D$ is redundant if it is supporting and attacking.*
- *A link $(s,t) \in L_D$ is dependent if it is neither supporting nor attacking.*

Example 4. *Consider again the ADF from Example 1. Note first that the acceptance condition of the statement a is a tautology and hence its truth value is independent of that of b; therefore (b,a) is a redundant link. The link (b,b) on the other hand is supporting, since whatever interpretation we start with (either that mapping b to $\boldsymbol{t}$ or that mapping b to $\boldsymbol{f}$), switching b to $\boldsymbol{t}$ clearly makes $\phi_b = b$ true. The link (b,b) is not attacking, since there is an interpretation that makes ϕ_b false (namely that mapping b to $\boldsymbol{f}$) and s.t. switching b to $\boldsymbol{t}$ makes ϕ_b true.*

The link (b,c) is supporting since $\phi_c = c \rightarrow b$ is true whenever b is mapped to true; it is not attacking since the interpretation mapping c to $\boldsymbol{t}$ and b to $\boldsymbol{f}$ makes ϕ_c false, while switching b from $\boldsymbol{f}$ to $\boldsymbol{t}$ makes ϕ_c true. The link (c,c) on the other hand is attacking since the only interpretation that makes $\phi_c = c \rightarrow b$ false is that mapping b to $\boldsymbol{f}$ and c to $\boldsymbol{t}$; hence switching c to $\boldsymbol{t}$ has no effect on the result. The link is not supporting since there is an interpretation mapping ϕ_c to $\boldsymbol{t}$ (i.e. the one mapping b to $\boldsymbol{f}$ and c to $\boldsymbol{f}$) s.t. switching c to $\boldsymbol{t}$ makes ϕ_c false.

For an example of a dependent link, assume the acceptance condition ϕ_c associated to c is $b \oplus c$ rather than $c \rightarrow b$. Then the link (b,c) would not be supporting since switching b to $\boldsymbol{t}$ for the interpretation mapping b to $\boldsymbol{f}$ and c to $\boldsymbol{t}$ would make the truth value of our new ϕ_c switch from $\boldsymbol{t}$ to $\boldsymbol{f}$. If the acceptance condition ϕ_c were $b \oplus c$, the link (b,c) would also not be attacking since switching b to $\boldsymbol{t}$ for the interpretation mapping b to $\boldsymbol{f}$ and c to $\boldsymbol{f}$ would make the truth value of ϕ_c go from $\boldsymbol{f}$ to $\boldsymbol{t}$. For analogous reasons, the link (c,c) for the new acceptance condition ϕ_c would also neither be attacking nor supporting.

Given an ADF D, we denote the supporting links by L_D^+, the attacking links by L_D^-, and those which are neither supporting nor attacking -, i.e. the dependent links,- as $L_D^\times$.

Computing the type of a link of an ADF is coNP-complete in general. It hence makes sense to further distinguish links according to whether their link type is known or unknown. For an ADF D we thus denote the links whose type is unknown by $L_D^?$, while those for which the types are known are represented via $L_D^!$ (i.e. $L_D^! := L_D \setminus L_D^?$). Therefore, for instance, the attacking links whose link type is unknown are $L_D^? \cap L_D^-$; the dependent links whose link type is known are $L_D^! \cap L_D^\times$.

The complexity results mentioned at the beginning of the current section follow from the fact that, as is expressed in Proposition 4, given an ADF $D = (S, \{\phi_s\}_{s \in S})$, a statement s of the ADF, and a three valued interpretation v, the number of completions one needs to check in order to verify that $w(\phi_s) = x$ for all $w \in [v]_2$ and some $x \in \{\mathbf{t}, \mathbf{f}\}$ is actually exponential in $k+l$ with $k = |L^{\times}_D|$ and $l = |L^{?}_D \cap (L^{-}_D \cup L^{+}_D)|$. This means that computing the result of the characteristic operator $\Gamma_D(v)(s)$ is also exponential in $k+l$.

As a result, if one considers a fixed constant $c \geq 0$ such that $k + l \leq c$ ($k = |L^{\times}_D|$, $l = |L^{?}_D \cap (L^{-}_D \cup L^{+}_D)|$), the complexity of computing $\Gamma_D(v)(s)$ is polynomial in c. I.e. if the number of links which are not known to be attacking and/or supporting is bounded by a fixed constant, the complexity of computing the result of the characteristic operator Γ_D, and hence one dimension of the complexity of computing the result of the different semantics, is reduced.

Proposition 4. *(Generalisation of Lemma 4.1.18 in [Wal14]) Let $D = (S, \{\phi_s\}_{s \in S})$ be an ADF, $s \in S$, and v a three valued interpretation. Let $c^{\mathbf{t}}_{D,v,s}$ and $c^{\mathbf{f}}_{D,v,s}$ be the interpretations on $par_D(s)$ defined as*

$$c^{\mathbf{t}}_{D,v,s}(s') := \begin{cases} \mathbf{t} \text{ if } v(s') = \mathbf{t} \\ \mathbf{f} \text{ if } v(s') = \mathbf{f} \\ \mathbf{t} \text{ if } v(s') = \mathbf{u} \text{ and } (s', s) \in L^{!}_D \cap L^{-}_D \\ \mathbf{f} \text{ if } v(s') = \mathbf{u} \text{ and } (s', s) \in L^{!}_D \cap L^{+}_D \\ \mathbf{u} \text{ if } v(s') = \mathbf{u} \text{ and } (s', s) \in L^{?}_D \cup L^{\times}_D \end{cases}$$

$$c^{\mathbf{f}}_{D,v,s}(s') := \begin{cases} \mathbf{t} \text{ if } v(s') = \mathbf{t} \\ \mathbf{f} \text{ if } v(s') = \mathbf{f} \\ \mathbf{f} \text{ if } v(s') = \mathbf{u} \text{ and } (s', s) \in L^{!}_D \cap L^{-}_D \\ \mathbf{t} \text{ if } v(s') = \mathbf{u} \text{ and } (s', s) \in L^{!}_D \cap L^{+}_D \\ \mathbf{u} \text{ if } v(s') = \mathbf{u} \text{ and } (s', s) \in L^{?}_D \cup L^{\times}_D \end{cases}$$

Then

1. *$w(\phi_s) = \mathbf{t}$ for all $w \in [v]_2$ iff $w'(\phi_s) = \mathbf{t}$ for all $w' \in [c^{\mathbf{t}}_{D,v,s}]_2$;*
2. *$w(\phi_s) = \mathbf{f}$ for all $w \in [v]_2$ iff $w'(\phi_s) = \mathbf{f}$ for all $w' \in [c^{\mathbf{f}}_{D,v,s}]_2$.*

Proof. The "only if" direction of the both items follow from the fact that $[c^{\mathbf{t}}_{D,v,s}]_2 \subseteq [v]_2$ and $[c^{\mathbf{f}}_{D,v,s}]_2 \subseteq [v]_2$ respectively.

As to the "if" direction for the first item, let w be a two valued interpretation on $par_D(s)$ and let $[w]^{\mathbf{t}}_{\Delta}$ be the set of two valued interpretations on $par_D(s)$ that are different from w only in that they either set attackers of s from false to true or supporters of s from true to false. I.e. if $w' \in [w]^{\mathbf{t}}_{\Delta}$ and $w'(s') \neq w(s')$ for some $s' \in par_D(s)$, then either $s' \in L^{+}_D$ and $w'(s') = \mathbf{f}$, or $(s', s) \in L^{-}_D$ and $w'(s') = \mathbf{t}$. It is proven in Lemma 4.1.18 in [Wal14] that

- Property J: If $w(\phi_s) = \mathbf{f}$, then $w'(\phi_s) = \mathbf{f}$ for any $w' \in [w]^{\mathbf{t}}_{\Delta}$[4].

Now, the first item of Proposition 4 follows from Property J and the fact that for any $w \in [v]_2$, there is a $w' \in [w]^{\mathbf{t}}_{\Delta}$ s.t. also $w' \in [c^{\mathbf{t}}_{D,v,s}]_2$ ("Observation A"). Indeed, consider w'' defined as $w''(s') = c^{\mathbf{t}}_{D,v,s}(s')$ if $s' \in L^{!}_{D} \cap (L^{-}_{D} \cup L^{+}_{D})$, but $w''(s') = w(s')$ otherwise. Then, by construction $w'' \in [w]^{\mathbf{t}}_{\Delta}$, but it is also easy to see that $w'' \in [c^{\mathbf{t}}_{D,v,s}]_2$. The latter is because w'' is (possibly) different from $c^{\mathbf{t}}_{D,v,s}$ on the statements $s' \in par_D(s)$ s.t. $v(s') = \mathbf{u}$ and $(s', s) \in L^{?}_{D} \cup L^{\times}_{D}$. And for such statements $c^{\mathbf{t}}_{D,v,s}(s') = \mathbf{u} \leq_i w''(s')$ whatever the value of $w''(s') = w(s')$.

Putting all the above together, we have that if $w(\phi_s) = \mathbf{f}$ for some $w \in [v]_2$, then by Property J $w'(\phi_s) = \mathbf{f}$ for any $w' \in [w]^{\mathbf{t}}_{\Delta}$. Thus by Observation A there is a $w' \in [w]^{\mathbf{t}}_{\Delta} \cap [c^{\mathbf{t}}_{D,v,s}]_2$ s.t. $w'(\phi_s) = \mathbf{f}$. In conclusion, if $w''(\phi_s) = \mathbf{t}$ for all $w'' \in [c^{\mathbf{t}}_{D,v,s}]_2$, then also $w(\phi_s) = \mathbf{t}$ for all $w \in [v]_2$ must be the case.

The "if direction" of the second item of Proposition 4 is proven in the same manner as the "if direction" of the first item, using a variant of Property J (also stated in [Wal14]) adequate to this case. ☐

We call the interpretations $c^{\mathbf{t}}_{D,v,s}$ and $c^{\mathbf{f}}_{D,v,s}$ from Proposition 4 the "canonical interpretations" (for D, s, and v). Proposition 4 is a (simple) generalisation of Lemma 4.1.18 in [Wal14]; see also a similar result for k-BADFs in the form of Lemma 4.1. in [LMN$^+$18a]. Corollary 2 is a straightforward consequence of Proposition 4.

Corollary 2. *Let $D = (S, \{\phi_s\}_{s \in S})$ be an ADF, $s \in S$, and v a three valued interpretation. Also, let $c^{\mathbf{t}}_{D,v,s}$ and $c^{\mathbf{f}}_{D,v,s}$ be the canonical interpretations on $par_D(s)$ defined as in Proposition 4. Then*

1. *$w(\phi_s) = \mathbf{t}$ for some $w \in [v]_2$ iff $w(\phi_s) = \mathbf{t}$ for some $w \in [c^{\mathbf{f}}_{D,v,s}]_2$;*
2. *$w(\phi_s) = \mathbf{f}$ for some $w \in [v]_2$ iff $w(\phi_s) = \mathbf{f}$ for some $w \in [c^{\mathbf{t}}_{D,v,s}]_2$.*

Proof. The first item follows from taking the contrapositive of both directions of the biconditional in the second item in Proposition 4, while the second item follows from taking the contrapositives of both directions of the first item in Proposition 4. ☐

While in Dung's AFs links between arguments represent attacks (as can be deduced from the representation of AFs as ADFs expressed in Definition 10), BADFs generalise AFs in allowing supporting as well as attacking links. The complexity of the reasoning tasks that are of interest to this work for ADFs which like Dung's AFs and BADFs have up to k hard links (dependent links or links that are not dependent, but whose link type

[4]Among the assumptions of Lemma 4.1.18 from [Wal14] are that D is bipolar and that D has no redundant links; but the proof of Property J does not make use of either of these assumptions.

σ	*adm*	*com*	*prf*	*grd*	*mod*	*stb*
$Cred_\sigma$	NP-c	NP-c	NP-c	in P	NP-c	NP-c
$Skept_\sigma$	trivial	in P	Π_2^P-c	in P	coNP-c	coNP-c
Ver_σ	in P	in P	coNP-c	in P	in P	in P

Table 3.1: Complexity results for ADFs such as AFs and BADFs, which have up to k hard links for some fixed constant $k \geq 0$. Follow from results in [LMN$^+$18a] and [Wal14, SW15].

is unknown) for some constant $k \geq 0$ are summarised in Table 3.1. They are based on Proposition 4 and are, again, generalisations of the results about k-BADFs and BADFs from [LMN$^+$18a] and [Wal14, SW15] respectively.

3.1.3.2 Encoding the types of the links of ADFs

In what remains of Section 3.1.3 we assume that an ADF is given together with a list of the type of the links of the ADF for all those links whose type is known. There may also be links whose type is unknown (see Section 3.1.3.1). Given such an ADF we will now develop an encoding that specifies the type of the links for those links whose type is known. Also, the module in question specifies how to determine the link types for those links whose type is unknown. For the latter links, the encoding pretty much recasts the definition of the various link types (Definition 16) in QBF terms.

To refer to attacking and supporting links within our encodings we will use (disjoint) sets of indexed variables A_{L_D} and O_{L_D} indexed with the links of the ADF of interest D ("A" for "attack" and "O" for "support"). We will also need variables P_S and P'_S indexed with the statements S of D in order to encode the definitions of attacking and supporting links as given in Definition 16. We remind the reader of our notation for propositional atoms indexed w.r.t. some set T_1 "projected" on a set of indices $T_2 \subseteq T_1$. Thus, for instance, $P_{par_D(s)} = \{p_{s'} \mid p_{s'} \in P_S \textit{ and } s' \in par_D(s)\}$ for any $s \in S$ (we refrain from further reminders about the notation for the projection on a subset of indices in what remains of this work).

The module specifying the link types of an ADF $D = (S, \{\phi_s\}_{s \in S})$ (together with information about the links of D) is as follows:

$$links[D, A_{L_D}, O_{L_D}, P_S, P'_S] := \chi_1 \wedge \chi_2 \wedge \chi_3 \wedge \chi_4 \wedge \chi_5$$

$$\chi_1 := \bigwedge_{(s_1,s_2) \in L_D^! \cap L_D^+} o_{(s_1,s_2)}$$

$$\chi_2 := \bigwedge_{(s_1,s_2) \in L_D^! \cap L_D^-} a_{(s_1,s_2)}$$

$$\begin{aligned}
\chi_3 &:= \bigwedge_{(s_1,s_2)\in L^!_D \cap L^\times_D} (\neg o_{(s_1,s_2)} \wedge \neg a_{(s_1,s_2)}) \\
\chi_4 &:= \bigwedge_{(s_1,s_2)\in L^?_D} (o_{(s_1,s_2)} \leftrightarrow \psi_{(s_1,s_2)}) \\
\chi_5 &:= \bigwedge_{(s_1,s_2)\in L^?_D} (a_{(s_1,s_2)} \leftrightarrow \psi'_{(s_1,s_2)}) \\
\psi_{(s_1,s_2)} &:= \forall P_{par_D(s_2)} \cup \{p'_{s_1}\}((\phi_{s_2}{}^{P_{par_D(s_2)}} \wedge p'_{s_1}) \rightarrow (\phi_{s_2}{}^{P_{par_D(s_2)}}[p_{s_1}/p'_{s_1}])) \\
\psi'_{(s_1,s_2)} &:= \forall P_{par_D(s_2)} \cup \{p'_{s_1}\}((\neg\phi_{s_2}{}^{P_{par_D(s_2)}} \wedge p'_{s_1}) \rightarrow (\neg\phi_{s_2}{}^{P_{par_D(s_2)}}[p_{s_1}/p'_{s_1}]))
\end{aligned}$$

Lemma 7 states the meaning of the module $links[.,.,.,.,.]$ for our encodings in formal terms.

Lemma 7. *Let $D = (S, \{\phi_s\}_{s\in S})$ be an ADF, $s_a \in S$, $s_b \in S$, and A_{L_D}, O_{L_D}, P_S, P'_S indexed atoms. Also, let $\Psi = links[D, A_{L_D}, O_{L_D}, P_S, P'_S]$. Then*

1. *$\hat{v}(o_{(s_a,s_b)}) = \boldsymbol{t}$ for any interpretation $\hat{v}$ s.t. $\hat{v} \models \Psi$ iff $(s_a, s_b) \in L^+_D$,*
2. *$\hat{v}(a_{(s_a,s_b)}) = \boldsymbol{t}$ for any interpretation $\hat{v}$ s.t. $\hat{v} \models \Psi$ iff $(s_a, s_b) \in L^-_D$, and*
3. *$\hat{v}(o_{(s_a,s_b)}) = \hat{v}(a_{(s_a,s_b)}) = \boldsymbol{f}$ for any interpretation $\hat{v}$ s.t. $\hat{v} \models \Psi$ iff $(s_a, s_b) \in L^\times_D$.*

Proof. (sketch) The reason for the truth of the first item is that on the one hand for any interpretation $\hat{v}$ s.t. $\hat{v} \models links[D, A_{L_D}, O_{L_D}, P_S, P'_S]$ we have that, (roughly) since $\hat{v} \models (\bigwedge_{(s_1,s_2)\in L^!_D \cap L^+_D} o_{(s_1,s_2)})$, $\hat{v}(o_{(s_a,s_b)}) = \mathbf{t}$ for $(s_a, s_b) \in L^!_D$ iff $(s_a, s_b) \in L^+_D$. On the other hand, for any interpretation $\hat{v}$ s.t. $\hat{v} \models links[D, A_{L_D}, O_{L_D}, P_S, P'_S]$ we have that also $\hat{v}(o_{(s_a,s_b)}) = \mathbf{t}$ for $(s_a, s_b) \notin L^!_D$ (i.e. $(s_a, s_b) \in L^?_D$) iff $(s_a, s_b) \in L^+_D$. The latter is due to the fact that if $\hat{v} \models links[D, A_{L_D}, O_{L_D}, P_S, P'_S]$, then $\hat{v} \models o_{(s_a,s_b)} \leftrightarrow \forall P_{par_D(s_b)} \cup \{p'_{s_a}\}((\phi_{s_b}{}^{P_{par_D(s_b)}} \wedge p'_{s_a}) \rightarrow (\phi_{s_b}{}^{P_{par_D(s_b)}}[p_{s_a}/p'_{s_a}]))$. By Lemma 1 this means that $\hat{v} \models o_{(s_a,s_b)}$ iff for all interpretations $\hat{u}$ it is the case that for $\hat{v}' := \hat{v}[(P_{par_D(s_2)} \cup \{p'_{s_1}\})/\hat{u}(P_{par_D(s_2)} \cup \{p'_{s_1}\})]$, $\hat{v}' \models ((\phi_{s_b}{}^{P_{par_D(s_b)}} \wedge p'_{s_a}) \rightarrow (\phi_{s_b}{}^{P_{par_D(s_b)}}[p_{s_a}/p'_{s_a}])$. The latter corresponds exactly to the requirement for $(s_a, s_b) \in L^+_D$ according to Definition 16. The proof of the second item is analogous to the proof of the first item. The third item follows directly from the first two items and the definition of $(s_a, s_b) \in L^\times_D$ (again, see Definition 16). □

We will later need to remove the inner quantifiers of the module encoding the link types of an ADF D, leaving us with the "matrix" of the module. For this we use extra variables indexed w.r.t. several sets which we define now. In the first place we define the sets of indices $< L^?_D, Q, 1 >:= \{s, (s_1, s_2), Q, T \mid s \in par_D(s_2), (s_1, s_2) \in L^?_D, T \in \{O, A\}\}$ and $< L^?_D, Q, 2 >:= \{s_1, (s_1, s_2), Q, T \mid (s_1, s_2) \in L^?_D, T \in \{O, A\}\}$ (the difference between the two sets of indices is in the first index). Here $Q \in \{\exists, \forall\}$. We also define a form of projection on the set of indices $< L^?_D, Q, 1 >$: $< L^?_D, Q, 1 >\downharpoonright_{(s_a,s_b),T} := \{s, (s_a, s_b), Q, T \mid s \in par_D(s_b)\}$. Here s_a and s_b are specific statements in S, $Q \in \{\exists, \forall\}$, and $T \in \{O, A\}$ ("O" once more stands for "support", while "A" stands for "attack").

The matrix of the module for the link types of an ADF D as considered all along is given below. It is obtained from $links[D, A_{L_D}, O_{L_D}, P_S, P'_S]$ via the equivalences in Proposition 1 (and the definition of the logical connective "$\leftrightarrow$" as well as removing quantifiers).

$$links^M[D, A_{L_D}, O_{L_D}, P_{<L^?_D,\exists,1>}, P'_{<L^?_D,\exists,2>}, P_{<L^?_D,\forall,1>}, P'_{<L^?_D,\forall,2>}] := X$$

$$X := \chi_1 \wedge \chi_2 \wedge \chi_3 \wedge \chi_4$$

$$\chi_1 := (\bigwedge_{(s_1,s_2) \in L^!_D \cap L^+_D} o_{(s_1,s_2)}) \wedge (\bigwedge_{(s_1,s_2) \in L^!_D \cap L^-_D} a_{(s_1,s_2)})$$

$$\chi_2 := \bigwedge_{(s_1,s_2) \in L^!_D \cap L^\times_D} (\neg o_{(s_1,s_2)} \wedge \neg a_{(s_1,s_2)})$$

$$\chi_3 := (\bigwedge_{(s_1,s_2) \in L^?_D} (o_{(s_1,s_2)} \rightarrow \psi_{(s_1,s_2)})) \wedge (\bigwedge_{(s_1,s_2) \in L^?_D} (\psi'_{(s_1,s_2)} \rightarrow o_{(s_1,s_2)}))$$

$$\chi_4 := (\bigwedge_{(s_1,s_2) \in L^?_D} (a_{(s_1,s_2)} \rightarrow \psi''_{(s_1,s_2)})) \wedge (\bigwedge_{(s_1,s_2) \in L^?_D} (\psi'''_{(s_1,s_2)} \rightarrow a_{(s_1,s_2)}))$$

$$\psi_{(s_1,s_2)} := \tau_1 \rightarrow \tau_2$$

$$\tau_1 := (\phi_{s_2}^{P_{<L^?_D,\forall,1>\downharpoonright_{(s_1,s_2),O}}} \wedge p'_{s_1,(s_1,s_2),\forall,O})$$

$$\tau_2 := (\phi_{s_2}^{P_{<L^?_D,\forall,1>\downharpoonright_{(s_1,s_2),O}}} [p_{s_1,(s_1,s_2),\forall,O} / p'_{s_1,(s_1,s_2),\forall,O}]))$$

$$\psi'_{(s_1,s_2)} := \tau_3 \rightarrow \tau_4$$

$$\tau_3 := (\phi_{s_2}^{P_{<L^?_D,\exists,1>\downharpoonright_{(s_1,s_2),O}}} \wedge p'_{s_1,(s_1,s_2),\exists,O})$$

$$\tau_4 := (\phi_{s_2}^{P_{<L^?_D,\exists,1>\downharpoonright_{(s_1,s_2),O}}} [p_{s_1,(s_1,s_2),\exists,O} / p'_{s_1,(s_1,s_2),\exists,O}]))$$

$$\psi''_{(s_1,s_2)} := \tau_5 \rightarrow \tau_6$$

$$\tau_5 := (\neg\phi_{s_2}^{P_{<L^?_D,\forall,1>\downharpoonright_{(s_1,s_2),A}}} \wedge p'_{s_1,(s_1,s_2),\forall,A})$$

$$\tau_6 := (\neg\phi_{s_2}^{P_{<L^?_D,\forall,1>\downharpoonright_{(s_1,s_2),A}}} [p_{s_1,(s_1,s_2),\forall,A} / p'_{s_1,(s_1,s_2),\forall,A}])$$

$$\psi'''_{(s_1,s_2)} := \tau_7 \rightarrow \tau_8$$

$$\tau_7 := (\neg\phi_{s_2}^{P_{<L^?_D,\exists,1>\downharpoonright_{(s_1,s_2),A}}} \wedge p'_{s_1,(s_1,s_2),\exists,A})$$

$$\tau_8 := (\neg\phi_{s_2}^{P_{<L^?_D,\exists,1>\downharpoonright_{(s_1,s_2),A}}} [p_{s_1,(s_1,s_2),\exists,A} / p'_{s_1,(s_1,s_2),\exists,A}])$$

Note that in the definition of the module $links^M[., ., ., ., ., ., .]$ (periods are placeholders for parameters of the module) we stretched the Definition of our notation ϕ^{P_T} for a propositional formula ϕ and an indexed set of propositional atoms P_T somewhat. E.g. ${\phi_{s_2}}^{P_{<L^?_D,\exists,1>\downharpoonleft(s_1,s_2),O}}$ means that every occurrence of a $s \in S$ in ϕ_{s_2} is replaced by the variable $p_{s,(s_1,s_2),\exists,O}$. Except if stated otherwise, in what remains of this chapter we stick with this convention. In other words, for complex indices the replacement of propositional atoms is always done considering the first element of the index.

3.1.3.3 Encodings for the admissible semantics

We now turn to the link information sensitive QBF encodings, starting with the encoding for the admissible semantics. To begin, we recall the defining encoding function for the admissible semantics from [Dil14], which quite closely follows the definition of this semantics as expressed in Definition 8.

Specifically, the "non link information sensitive" encoding $\mathcal{E}_{adm}$ (see below) involves three modules; the module $coh[.]$ to refer to a candidate three valued interpretation, the module $\leq_i [.,.]$ to refer to the completions of the candidate interpretation, and finally the module $eval[.,.,.]$ to express the condition that a candidate interpretation needs to fulfill for being admissible. The latter condition for an ADF $D = (S, \{\phi_s\}_{s\in S})$ and a candidate interpretation v is that if $v(s) = \mathbf{t}$ then for every completion $w \in [v]_2$ it must be the case that $w(\phi_s) = \mathbf{t}$, while if $v(s) = \mathbf{f}$ then for every $w \in [v]_2$, $w(\phi_s) = \mathbf{f}$ must hold.

$$
\begin{aligned}
\mathcal{E}_{adm}[D, P^3_S, P_S] &:= coh[P^3_S] \wedge \forall P_S(\leq_i [P^3_S, P_S] \rightarrow eval[D, P^3_S, P_S]) \\
eval[D, P^3_S, P_S] &:= \bigwedge_{s\in S} (({p_s}^{\oplus} \rightarrow {\phi_s}^{P_S}) \wedge ({p_s}^{\ominus} \rightarrow \neg{\phi_s}^{P_S}))
\end{aligned}
$$

Example 5. *The following is an encoding for credulous reasoning w.r.t. the admissible semantics for the statement a and the ADF D from Example 1. It is based on the defining encoding function $\mathcal{E}_{adm}$ from [Dil14] (and obtained via Proposition 2 and the equivalences in Proposition 1). Here $P^3_S = \{{p_a}^{\oplus}, {p_a}^{\ominus}, {p_b}^{\oplus}, {p_b}^{\ominus}, {p_c}^{\oplus}, {p_c}^{\ominus}\}$ and $P_S = \{p_a, p_b, p_c\}$.*

$$
\begin{aligned}
Cred_{adm}(D, a) &\equiv \exists P^3_S \forall P_S(\psi \wedge {p_a}^{\oplus}) \\
\psi &= coh[P^3_S] \wedge (\leq_i [P^3_S, P_S] \rightarrow eval[D, P^3_S, P_S]) \\
coh[P^3_S] &= \neg({p_a}^{\oplus} \wedge {p_a}^{\ominus}) \wedge \neg({p_b}^{\oplus} \wedge {p_b}^{\ominus}) \wedge \neg({p_c}^{\oplus} \wedge {p_c}^{\ominus}) \\
\leq_i [P^3_S, P_S] &= \psi_a \wedge \psi_b \wedge \psi_c \\
\psi_a &= (({p_a}^{\oplus} \rightarrow p_a) \wedge ({p_a}^{\ominus} \rightarrow \neg p_a)) \\
\psi_b &= (({p_b}^{\oplus} \rightarrow p_b) \wedge ({p_b}^{\ominus} \rightarrow \neg p_b)) \\
\psi_c &= (({p_c}^{\oplus} \rightarrow p_c) \wedge ({p_c}^{\ominus} \rightarrow \neg p_c)) \\
eval[D, P^3_S, P_S] &= \psi'_a \wedge \psi'_b \wedge \psi'_c \\
\psi'_a &= (({p_a}^{\oplus} \rightarrow (p_b \vee \neg p_b)) \wedge ({p_a}^{\ominus} \rightarrow \neg(p_b \vee \neg p_b)))
\end{aligned}
$$

$$\psi'_b = ((p_b{}^{\oplus} \rightarrow p_b) \wedge (p_b{}^{\ominus} \rightarrow \neg p_b))$$
$$\psi'_c = ((p_c{}^{\oplus} \rightarrow (p_c \rightarrow p_b)) \wedge (p_c{}^{\ominus} \rightarrow \neg(p_c \rightarrow p_b)))$$

The difference between the non link information sensitive defining encoding $\mathcal{E}_{adm}$ and the new link information sensitive encoding $\mathcal{E}'_{adm}$ we present now is that the new encoding reflects the fact that, as we explained in detail in Section 3.1.3.1, when information about link types of an ADF is present computing the result of evaluating the different semantics on ADFs can be simplified. Just as for the remaining semantics considered in this work, also for the admissible semantics this concretely means that it is not necessary to compute the result of evaluating each of the acceptance conditions under all completions of a candidate ADF interpretation. Rather, "only" a number of completions that is exponential in the number of links with dependent or unknown type (what we called hard links in Section 3.1.3.1) need to be considered per statement of the ADF.

Thus, the link information sensitive encoding for the admissible semantics is structurally very similar to the non link information sensitive encoding but now, first of all, includes the $links[.,.,.,.,.]$ module we defined in Section 3.1.3.2. Secondly, the modules $\leq_i [.,.]$ and $eval[.,.,.]$ in the non link information sensitive encoding are replaced with modules $canon[.,.,.,.,.]$ and $evalCanon[.,.,.]$, which make use of the link information captured via the $links[.,.,.,.,.]$ module and reflect the need for evaluating the acceptance conditions on the completions of the canonical completions only (see explanation in Section 3.1.3.1). In order to capture the completions of the canonical completions needed per statement for a set of statements S of an ADF D, we need atoms $R_{<S,par_D(S))>}$, indexed by the set $< S, par_D(S)) >:= \{s_1, s_2 \mid s_1 \in S, s_2 \in par_D(s_1)\}$, i.e. there is an atom for each parent of every statement in the ADF. Again, we define a form of projection on the indices $< s_a, par_D(s_a)) >:= \{s_a, s_b \mid s_b \in par_D(s_a)\}$ for a specific $s_a \in S$.

$$\begin{aligned}
\mathcal{E}'_{adm}[D, P^3_S, A_{L_D}, O_{L_D}, P_S, P'_S, R_{<S,par_D(S))>}] &:= \psi \wedge \psi' \\
\psi &:= coh[P^3_S] \wedge links[D, A_{L_D}, O_{L_D}, P_S, P'_S] \\
\psi' &:= \forall R_{<S,par_D(S))>}\ \psi'' \\
\psi'' &:= (canon[D, A_{L_D}, O_{L_D}, P^3_S, R_{<S,par_D(S))>}] \rightarrow evalCanon[D, P^3_S, R_{<S,par_D(S))>}])
\end{aligned}$$

The module $canon[.,.,.,.,.]$ expresses the completions of the canonical interpretations needed per statement as expressed in Proposition 4 and is defined as follows:

$$\begin{aligned}
canon[D, A_{L_D}, O_{L_D}, P^3_S, R_{<S,par_D(S))>}] &:= \bigwedge_{s_1 \in S, s_2 \in par_D(s_1)} (\psi_{s_1,s_2} \wedge \psi'_{s_1,s_2}) \\
\psi_{s_1,s_2} &:= (p_{s_2}{}^{\oplus} \rightarrow r_{s_1,s_2}) \wedge (p_{s_2}{}^{\ominus} \rightarrow \neg r_{s_1,s_2}) \\
\psi'_{s_1,s_2} &:= ((\neg p_{s_2}{}^{\oplus} \wedge \neg p_{s_2}{}^{\ominus}) \rightarrow (\chi_{s_1,s_2} \wedge \chi'_{s_1,s_2} \wedge \chi''_{s_1,s_2} \wedge \chi'''_{s_1,s_2}))
\end{aligned}$$

$$\begin{aligned}
\chi_{s_1,s_2} &:= ((p_{s_1}{}^{\oplus} \land o_{(s_2,s_1)}) \rightarrow \neg r_{s_1,s_2})\\
\chi'_{s_1,s_2} &:= ((p_{s_1}{}^{\oplus} \land \neg o_{(s_2,s_1)} \land a_{(s_2,s_1)}) \rightarrow r_{s_1,s_2})\\
\chi''_{s_1,s_2} &:= ((p_{s_1}{}^{\ominus} \land o_{(s_2,s_1)}) \rightarrow r_{s_1,s_2})\\
\chi'''_{s_1,s_2} &:= ((p_{s_1}{}^{\ominus} \land \neg o_{(s_2,s_1)} \land a_{(s_2,s_1)}) \rightarrow \neg r_{s_1,s_2})
\end{aligned}$$

Note that in the module *canon*[., ., ., ., .] completions of both canonical interpretations $c^{\mathbf{t}}_{D,v,s_1}$ and $c^{\mathbf{f}}_{D,v,s_1}$ from Proposition 4 needed per statement $s_1 \in S$ (for a candidate interpretation v for the ADF D) are specified. In effect, χ_{s_1,s_2} and χ'_{s_1,s_2} are needed for specifying the completions of the canonical interpretation $c^{\mathbf{t}}_{D,v,s_1}$ (of a candidate interpretation v for which $v(s_1) = \mathbf{t}$), while χ''_{s_1,s_2} and χ''_{s_1,s_2} are required for specifying the completions of the canonical interpretation $c^{\mathbf{f}}_{D,v,s_1}$ (here the assumption is that $v(s_1) = \mathbf{f}$). For later encodings we will need to separate the specification of both canonical completions, but not doing so for the admissible semantics allows for a slightly more compact defining encoding function for this semantics.

The definition of the module *evalCanon*[., ., .], on the other hand, is:

$$\begin{aligned}
&\mathit{evalCanon}[D,P^3_S,R_{<S,par_D(S))>}] := \psi\\
&\qquad \psi := \bigwedge_{s\in S} ((p_s{}^{\oplus} \rightarrow \phi_s{}^{R_{<s,par_D(s))>}}) \land (p_s{}^{\ominus} \rightarrow \neg\phi_s{}^{R_{<s,par_D(s))>}}))
\end{aligned}$$

As hinted at previously, this module is very similar to the module *eval*[., ., .] used in the non link information sensitive encoding but takes in account the completions of the canonical interpretations (as computed by the module *canon*[., ., ., ., .]) only.

Proposition 5. *Given an ADF $D = (S, \{\phi_s\}_{s\in S})$ and indexed variables P^3_S, A_{L_D}, O_{L_D},P_S, P'_S, $R_{<S,par_D(S))>}$, let $\mathcal{E}'_{adm}$ be the function returning the QBF with free variables in P^3_S, $\mathcal{E}'_{adm}[D, P^3_S, A_{L_D}, O_{L_D}, P_S, P'_S, R_{<S,par_D(S))>}]$. Then $\mathcal{E}'_{adm}$ is a defining encoding function for the admissible semantics.*

Proof. (sketch) Coherence of $\mathcal{E}'_{adm}$ follows from Lemma 4 and the fact that if $\hat{v} \models \psi$, where $\psi = \mathcal{E}'_{adm}[D, P^3_S, A_{L_D}, O_{L_D}, P_S, P'_S, R_{<S,par_D(S))>}]$, it is also the case that $\hat{v} \models coh[P^3_S]$.

For soundness, assume that $\hat{v} \models \mathcal{E}'_{adm}[P^3_S, A_{L_D}, O_{L_D}, P_S, P'_S, R_{<S,par_D(S))>}]$ and let v be a three valued interpretation s.t. $\hat{v} \cong_{P^3_S} v$. In particular, $\hat{v} \models links[D, A_{L_D}, O_{L_D}, P_S, P'_S]$. By Lemma 7 this means that the types of the links of D are correctly captured by the valuation of $\hat{v}$ on the variables in $A_{L_D} \cup O_{L_D}$. Also, $\hat{v} \models \forall R_{<S,par_D(S))>}(\psi' \rightarrow \psi'')$ with $\psi' = canon[D, A_{L_D}, O_{L_D}, P^3_S, R_{<S,par_D(S))>}]$ and $\psi'' = evalCanon[D, P^3_S, R_{<S,par_D(S))>}])$. By Lemma 1 this means that $\hat{z} \models \psi' \rightarrow \psi''$ for $\hat{z} := \hat{v}[R_{<S,par_D(S))>}/\hat{w}(R_{<S,par_D(S))>})]$ where $\hat{w}$ is any two valued interpretation . In particular, $\hat{z}' \models \psi' \rightarrow \psi''$ where $\hat{z}' := \hat{v}[R_{<S,par_D(S))>}/\hat{w}'(R_{<S,par_D(S))>})]$ and $\hat{w}'$ coincides with a completion of the canonical completion $c^{\mathbf{t}}_{D,v,s}$ of v (see Proposition 4) on the variables $R_{<s,par_D(s))>}$ for every statement $s \in S$ if $v(s) = \mathbf{t}$. On the other hand, $\hat{w}'$ coincides with a completion w' of the canonical completion $c^{\mathbf{f}}_{D,v,s}$ of v on the variables $R_{<s,par_D(s))>}$ for every $s \in S$ if $v(s) = \mathbf{f}$. Here, by $\hat{w}'$ "coinciding" with w' on $R_{<s,par_D(s))>}$ we mean that $\hat{w}'(r_{s,s'}) = \mathbf{t}$ iff $w(s') = \mathbf{t}$ for every $s' \in par_D(s)$. By construction we have that $\hat{z}' \cong_{P^3_S} v$, the types of the links of D are also correctly captured by the valuation of $\hat{z}'$ on the variables in $A_{L_D} \cup O_{L_D}$ and, finally, $\hat{z}'$ also coincides with the completion w' of v for every $s \in S$. It is also straightforward to prove that then $\hat{z}' \models canon[D, A_{L_D}, O_{L_D}, P^3_S, R_{<S,par_D(S))>}]$, which implies that also $\hat{z}' \models evalCanon[D, P^3_S, R_{<S,par_D(S))>}]$. It is also easy to prove that from $\hat{z}' \models evalCanon[D, P^3_S, R_{<S,par_D(S))>}]$ it follows that $v(s) \leq_i w'(\phi_s)$ for every $s \in S$. Now, since the completion w' was arbitrary we have that $v(s) \leq_i w''(\phi_s)$ for every $w'' \in [c^{\mathbf{t}}_{D,v,s}]_2$ when $v(s) = \mathbf{t}$, and $v(s) \leq_i w''(\phi_s)$ for every $w'' \in [c^{\mathbf{f}}_{D,v,s}]_2$ when $v(s) = \mathbf{f}$. Thus, by Proposition 4 and Definition 8, it follows that $v \in adm(D)$.

The proof for completeness is analogous to soundness but starting with a three valued interpretation $v \in adm(D)$ and in inverse form to the proof of soundness (using the same lemmas and propositions), constructing a two valued interpretation $\hat{v}$ s.t. $\hat{v} \cong_{P^3_S} v$ and $\hat{v} \models \mathcal{E}'_{adm}[D, P^3_S, A_{L_D}, O_{L_D}, P_S, P'_S, R_{<S,par_D(S))>}]$. □

Turning to the encodings for the reasoning tasks for the admissible semantics, we start by recalling that reasoning about the skeptical acceptance of a statement w.r.t. the admissible semantics is trivial, the reason being that the interpretation mapping every statement to the truth value $\mathbf{u}$ is always admissible. Hence, $Skept_{adm}(D, s^*) \equiv \bot$ for any ADF D and statement s^*. As to the encodings for the verification problem and credulous reasoning, these will be based on our link information sensitive defining encoding function $\mathcal{E}'_{adm}$ (via Proposition 2, the equivalences from Proposition 1, and the definition of the logical connective "$\rightarrow$").

The verification problem for an ADF D, an interpretation v, and the admissible semantics can be encoded as follows:

$$
\begin{aligned}
&Ver_{adm}(D, v) \equiv\ \exists\ V\ \forall\ V'(\psi \wedge \psi')[P^3_S/v(S)] \\
&\quad V := A_{L_D} O_{L_D} P_{<L^?_D,\exists,1>} P'_{<L^?_D,\exists,2>} \\
&\quad V' := P_{<L^?_D,\forall,1>} P'_{<L^?_D,\forall,2>} R_{<S,par_D(S))>} \\
&\quad \psi := coh[P^3_S] \wedge links^M[D, A_{L_D}, O_{L_D}, P_{<L^?_D,\exists,1>}, P'_{<L^?_D,\exists,2>}, P_{<L^?_D,\forall,1>}, P'_{<L^?_D,\forall,2>}] \\
&\quad \psi' := (canon[D, A_{L_D}, O_{L_D}, P^3_S, R_{<S,par_D(S))>}] \rightarrow evalCanon[D, P^3_S, R_{<S,par_D(S))>}])
\end{aligned}
$$

We note that when $L^?_D = \emptyset$ (i.e. $L^!_D = L_D$), then the encoding is reduced to the following form:

$$Ver_{adm}(D, v) \equiv\ \exists A_{L_D} O_{L_D} \forall\ R_{<S,par_D(S))>}(\psi \wedge \psi')[P^3_S/v(S)]$$

where ψ and ψ' are defined as above. The crucial difference is that when $L^?_D = \emptyset$ the sets of variables $P_{<L^?_D,\exists,1>}$, $P'_{<L^?_D,\exists,2>}$, $P_{<L^?_D,\forall,1>}$, and $P'_{<L^?_D,\forall,2>}$ needed to compute the link types for those links whose type is unknown are empty (see the definition of the $links^M[.,.,.,.,.,.,.]$ module in Section 3.1.3.2) . The consequence is that the quantifier blocks in the prefix of the encoding are simplified ($V = A_{L_D}O_{L_D}$ and $V' = R_{<S,par_D(S))>}$, rather than $V = A_{L_D}O_{L_D}P_{<L^?_D,\exists,1>}P'_{<L^?_D,\exists,2>}$ and $V' = P_{<L^?_D,\forall,1>}P'_{<L^?_D,\forall,2>}R_{<S,par_D(S))>}$).

In fact, when $L^?_D = \emptyset$ the encoding can be further simplified by replacing the variables in $A_{L_D} \cup O_{L_D}$ with the truth constants $\bot$ and $\top$ in accordance with what is known about the link types of the ADF. To write this down we introduce the notation $\psi[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)]$ for a QBF ψ with free variables in $A_{L^!_D} \cup O_{L^!_D}$. Here each $a_l \in A_{L^!_D}$ in ψ is replaced with $\top$ whenever $l \in L^!_D \cap L^-_D$ and with $\bot$ whenever $l \in L^!_D$ but $l \notin L^-_D$. In the same manner, each $o_l \in O_{L^!_D}$ in ψ is replaced with $\top$ whenever $l \in L^!_D \cap L^+_D$ and with $\bot$ whenever $l \in L^!_D$ but $l \notin L^+_D$. Note that the replacement of the variables in $A_{L_D} \cup O_{L_D}$ within the QBF ψ is partial, depending on the information that is available about the link types of the ADF. This information is represented symbolically by the function $\iota_D(L_D) : (A_{L^!_D} \cup O_{L^!_D}) \mapsto \{\bot, \top\}$.

So, when $L^?_D = \emptyset$ we have that

$$Ver_{adm}(D, v) \equiv \forall\ R_{<S,par_D(S))>}(\psi \wedge \psi')[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]$$

i.e. the first quantifier block has disappeared (since, now $V = \emptyset$). This leaves us with a QBF of prefix type Π_1, thus matching the complexity of the verification problem for the admissible semantics (coNP-c).

Moreover, when $L^?_D = \emptyset$ and, in addition, all links of the ADF are either attacking or supporting (i.e. the ADF is a BADF), truth values for all the remaining variables in V' (i.e. $R_{<S,par_D(S))>}$) can be derived from the truth values of the variables in $A_{L^!_D} \cup O_{L^!_D}$ and the interpretation v. To denote this we extend the notation $\psi[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]$ for a QBF ψ with variables in $P^3_S \cup A_{L^!_D} \cup O_{L^!_D}$ to denote the "propagation" of the information given by $\iota_D(L_D)$ and v also to a set of variables $R_{<S,par_D(S))>}$ indexed by $< S, par_D(S)) >$.

The latter is denoted by $\psi\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\}\}^{R_{<S,par_D(S))>}}$. The meaning of this notation is that every $r_{s_1,s_2} \in R_{<S,par_D(S))>}$ within ψ is replaced with $\top$ whenever either (i) $v(s_2) = \mathbf{t}$, (ii) $v(s_1) = \mathbf{t}$, $v(s_2) = \mathbf{u}$, $(s_2, s_1) \in L^!_D$, $(s_2, s_1) \in L^-_D$, $(s_2, s_1) \notin L^+_D$, or (iii) $v(s_1) = \mathbf{f}$, $v(s_2) = \mathbf{u}$, $(s_2, s_1) \in L^!_D$, $(s_2, s_1) \in L^+_D$. Analogously, every $r_{s_1,s_2} \in R_{<S,par_D(S))>}$ is replaced with $\bot$ whenever either (i) $v(s_2) = \mathbf{f}$, (ii) $v(s_1) = \mathbf{t}$, $v(s_2) = \mathbf{u}$, $(s_2, s_1) \in L^!_D$, $(s_2, s_1) \in L^+_D$, or (iii) $v(s_1) = \mathbf{f}$, $(s_2, s_1) \in L^!_D$, $(s_2, s_1) \in L^-_D$, $(s_2, s_1) \notin L^+_D$. In other words, $\psi\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\}\}^{R_{<S,par_D(S))>}}$ replaces each variable $r_{s_1,s_2} \in R_{<S,par_D(S))>}$ for which $(s_s, s_1) \in L^!_D \cap (L^-_D \cup L^+_D)$ with $\bot$ and $\top$ in accordance with the truth value assigned to s_2 by the canonical interpretations $c^{\mathbf{t}}_{D,v,s_1}$ or $c^{\mathbf{f}}_{D,v,s_1}$ from Proposition 4 whenever $v(s_1) = \mathbf{t}$ or $v(s_1) = \mathbf{f}$ respectively.

In the situation that the ADF D is a BADF, applying the replacements defined by $\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\}\}^{R_{<S,par_D(S))>}}$ to our encoding results in all the variables in $R_{<S,par_D(S))>}$ to be replaced with the truth constants $\bot$ and $\top$. We thus have that

$$Ver_{adm}(D, v) \equiv\ (\psi \wedge \psi')\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\}\}^{R_{<S,par_D(S))>}}$$

i.e. we are left with a propositional formula without variables, only with truth constants. Evaluating such a propositional formula can be done in polynomial time and thus matches the complexity of the verification problem for the admissible semantics when the ADF is bipolar. It is in the precise sense developed in this and the previous paragraphs that we say that our encoding is link information sensitive (see Example 6 for further discussion for the encoding for credulous reasoning w.r.t. the admissible semantics).

We turn now to encoding credulous reasoning for the admissible semantics in prenex normal form. Given an ADF D and a statement s^* of the ADF, it is as follows:

$$\begin{aligned}
&Cred_{adm}(D, s^*) \equiv\ \exists\ V\ \forall\ V'(\psi \wedge \psi' \wedge p_{s^*}{}^{\oplus})\\
V &:= A_{L_D} O_{L_D} P_{<L^?_D,\exists,1>} P'_{<L^?_D,\exists,2>} P^3_S\\
V' &:= P_{<L^?_D,\forall,1>} P'_{<L^?_D,\forall,2>} R_{<S,par_D(S))>}\\
\psi &:= coh[P^3_S] \wedge links^M[D, A_{L_D}, O_{L_D}, P_{<L^?_D,\exists,1>}, P'_{<L^?_D,\exists,2>}, P_{<L^?_D,\forall,1>}, P'_{<L^?_D,\forall,2>}]\\
\psi' &:= (canon[D, A_{L_D}, O_{L_D}, P^3_S, R_{<S,par_D(S))>}] \rightarrow evalCanon[D, P^3_S, R_{<S,par_D(S))>}])
\end{aligned}$$

The encoding has prefix type Σ_2, thus matching the complexity of credulous reasoning w.r.t. the admissible semantics which is Σ^P_2-complete.

If the types of all of the links of the ADF D are known (i.e. $L^!_D = L_D$), then the encoding can be simplified to

$$Cred_{adm}(D, s^*) \equiv\ \exists\ A_{L_D} O_{L_D} P^3_S\ \forall\ R_{<S,par_D(S))>}(\psi \wedge \psi' \wedge p_{s^*}{}^{\oplus})$$

i.e. the extra variables needed for computing the types of the links whose type is unknown are not needed.

Even if the ADF D is bipolar (i.e. $L_D = L_D^! \cap (L_D^- \cup L_D^+)$), we cannot remove the first quantifier block as the variables P_S^3 are needed to compute a "candidate" interpretation for being admissible. Nevertheless, given a guess $v^?$ for such a candidate interpretation the whole encoding can be simplified to

$$Cred_{adm}(D, s^*) \equiv \ (\psi \wedge \psi' \wedge p_{s^*}{}^{\oplus})\{\{[A_{L_D^!} \cup O_{L_D^!}/\iota_D(L_D)][P_S^3/v^?(P_S^3)]\}\}^{R_{<S, par_D(S))>}}$$

which is a propositional formula only having truth constants and whose satisfiability can, therefore, be decided in polynomial time. In other words, having a guess for the variables P_S^3 (and known or "guessed" values for the variables in $A_{L_D^!} \cup O_{L_D^!}$), also the truth values for all variables $R_{<S, par_D(S))>}$ are determined. This reflects the complexity of credulous reasoning for an ADF D and statement s^* w.r.t. the admissible semantics for BADFs which is NP-complete, the main source of complexity coming from guessing an interpretation v, with checking for admissibility of v and for $v(s^*) = \mathbf{t}$ being the easier sub-tasks.

We give an example of the encoding for credulous reasoning for a specific ADF in Example 6. We also explore there in more detail the a-priori advantages and disadvantages of the link information sensitive vs. the non link information sensitive encodings. The results of an empirical evaluation on the extent to which the expected advantages and disadvantages are reflected in the performance of an implementation of the encodings can be found in Appendix A (discussion in Section 3.1.4).

Example 6. *The following is an encoding for credulous reasoning w.r.t. the admissible semantics for the statement a and the ADF D from Example 1. It is based on the link information sensitive defining encoding function $\mathcal{E}'_{adm}$. We assume here that the only link whose type is unknown is (b, c) (i.e. $L_D^? = \{(b, c)\}$), while $L_D^! = \{(b, a), (b, b), (c, c)\}$). For the encoding we use the following sets of indexed propositional atoms:*

$$\begin{aligned}
P_S^3 &= \{p_a{}^{\oplus}, p_a{}^{\ominus}, p_b{}^{\oplus}, p_b{}^{\ominus}, p_c{}^{\oplus}, p_c{}^{\ominus}\},\\
A_{L_D} &= \{a_{(b,a)}, a_{(b,b)}, a_{(b,c)}, a_{(c,c)}\},\\
O_{L_D} &= \{o_{(b,a)}, o_{(b,b)}, o_{(b,c)}, o_{(c,c)}\},\\
P_{<L_D^?, \exists, 1>} &= \{p_{b,(b,c),\exists,O}, p_{c,(b,c),\exists,O}, p_{b,(b,c),\exists,A}, p_{c,(b,c),\exists,A}\},\\
P_{<L_D^?, \exists, 2>} &= \{p'_{b,(b,c),\exists,O}, p'_{b,(b,c),\exists,A}\},\\
P_{<L_D^?, \forall, 1>} &= \{p_{b,(b,c),\forall,O}, p_{c,(b,c),\forall,O}, p_{b,(b,c),\forall,A}, p_{c,(b,c),\forall,A}\},
\end{aligned}$$

$$\begin{aligned} P_{<L^?_D,\forall,2>} &= \{p'_{b,(b,c),\forall,O}, p'_{b,(b,c),\forall,A}\},\\ R_{<S,par_D(S))>} &= \{r_{a,b}, r_{b,b}, r_{c,b}, r_{c,c}\}. \end{aligned}$$

The encoding then is as follows:

$$\begin{aligned} &Cred_{adm}(D,a) \equiv \exists\, V\ \forall\, V'(\psi \wedge \psi' \wedge p_a{}^{\oplus})\\ &\quad V := A_{L_D} O_{L_D} P_{<L^?_D,\exists,1>} P'_{<L^?_D,\exists,2>} P^3_S\\ &\quad V' := P_{<L^?_D,\forall,1>} P'_{<L^?_D,\forall,2>} R_{<S,par_D(S))>}\\ &\quad \psi := coh[P^3_S] \wedge links^M[D, A_{L_D}, O_{L_D}, P_{<L^?_D,\exists,1>}, P'_{<L^?_D,\exists,2>}, P_{<L^?_D,\forall,1>}, P'_{<L^?_D,\forall,2>}]\\ &\quad \psi' := (canon[D, A_{L_D}, O_{L_D}, P^3_S, R_{<S,par_D(S))>}] \rightarrow evalCanon[D, P^3_S, R_{<S,par_D(S))>}]) \end{aligned}$$

The modules used within the encoding are defined next:

$$\begin{aligned} coh[P^3_S] &= \neg(p_a{}^{\oplus} \wedge p_a{}^{\ominus}) \wedge \neg(p_b{}^{\oplus} \wedge p_b{}^{\ominus}) \wedge \neg(p_c{}^{\oplus} \wedge p_c{}^{\ominus})\\ links^M[D, A_{L_D}, O_{L_D}, P_{<L^?_D,\exists,1>}, P'_{<L^?_D,\exists,2>}, P_{<L^?_D,\forall,1>}, P'_{<L^?_D,\forall,2>}] &= \chi_1 \wedge \chi_2\\ \chi_1 &= (a_{(b,a)} \wedge o_{(b,a)} \wedge o_{(b,b)} \wedge \neg a_{(b,b)} \wedge \neg o_{(c,c)} \wedge a_{(c,c)})\\ \chi_2 &= \chi_2' \wedge \chi_2'' \wedge \chi_2''' \wedge \chi_2''''\\ \chi_2' &= (o_{(b,c)} \rightarrow (\tau_1 \rightarrow \tau_2))\\ \chi_2'' &= ((\tau_3 \rightarrow \tau_4) \rightarrow o_{(b,c)})\\ \chi_2''' &= (a_{(b,c)} \rightarrow (\tau_5 \rightarrow \tau_6))\\ \chi_2'''' &= ((\tau_7 \rightarrow \tau_8) \rightarrow a_{(b,c)})\\ \tau_1 &= ((p_{c,(b,c),\forall,O} \rightarrow p_{b,(b,c),\forall,O}) \wedge p'_{b,(b,c),\forall,O})\\ \tau_2 &= (p_{c,(b,c),\forall,O} \rightarrow p'_{b,(b,c),\forall,O})\\ \tau_3 &= ((p_{c,(b,c),\exists,O} \rightarrow p_{b,(b,c),\exists,O}) \wedge p'_{b,(b,c),\exists,O})\\ \tau_4 &= (p_{c,(b,c),\exists,O} \rightarrow p'_{b,(b,c),\exists,O})\\ \tau_5 &= (\neg(p_{c,(b,c),\forall,A} \rightarrow p_{b,(b,c),\forall,A}) \wedge p'_{b,(b,c),\forall,A})\\ \tau_6 &= \neg(p_{c,(b,c),\forall,A} \rightarrow p'_{b,(b,c),\forall,A})\\ \tau_7 &= (\neg(p_{c,(b,c),\exists,A} \rightarrow p_{b,(b,c),\exists,A}) \wedge p'_{b,(b,c),\exists,A})\\ \tau_8 &= \neg(p_{c,(b,c),\exists,A} \rightarrow p'_{b,(b,c),\exists,A})\\ canon[D, A_{L_D}, O_{L_D}, P^3_S, R_{<S,par_D(S))>}] &= \bigwedge_{s_1 \in S, s_2 \in par_D(s_1)} (\psi_{s_1,s_2} \wedge \psi'_{s_1,s_2}) \end{aligned}$$

$$
\begin{aligned}
\psi_{a,b} &= (p_b{}^{\oplus} \to r_{a,b}) \wedge (p_b{}^{\ominus} \to \neg r_{a,b})\\
\psi'_{a,b} &= ((\neg p_b{}^{\oplus} \wedge \neg p_b{}^{\ominus}) \to (\omega_{a,b} \wedge \omega'_{a,b} \wedge \omega''_{a,b} \wedge \omega'''_{a,b}))\\
\omega_{a,b} &= ((p_a{}^{\oplus} \wedge o_{(b,a)}) \to \neg r_{a,b})\\
\omega'_{a,b} &= ((p_a{}^{\oplus} \wedge \neg o_{(b,a)} \wedge a_{(b,a)}) \to r_{a,b})\\
\omega''_{a,b} &= ((p_a{}^{\ominus} \wedge o_{(b,a)}) \to r_{a,b})\\
\omega'''_{a,b} &= ((p_a{}^{\ominus} \wedge \neg o_{(b,a)} \wedge a_{(b,a)}) \to \neg r_{a,b})\\
\psi_{b,b} &= (p_b{}^{\oplus} \to r_{b,b}) \wedge (p_b{}^{\ominus} \to \neg r_{b,b})\\
\psi'_{b,b} &= ((\neg p_b{}^{\oplus} \wedge \neg p_b{}^{\ominus}) \to (\omega_{b,b} \wedge \omega'_{b,b} \wedge \omega''_{b,b} \wedge \omega'''_{b,b}))\\
\omega_{b,b} &= ((p_b{}^{\oplus} \wedge o_{(b,b)}) \to \neg r_{b,b})\\
\omega'_{b,b} &= ((p_b{}^{\oplus} \wedge \neg o_{(b,b)} \wedge a_{(b,b)}) \to r_{b,b})\\
\omega''_{b,b} &= ((p_b{}^{\ominus} \wedge o_{(b,b)}) \to r_{b,b})\\
\omega'''_{b,b} &= ((p_b{}^{\ominus} \wedge \neg o_{(b,b)} \wedge a_{(b,b)}) \to \neg r_{b,b})\\
\psi_{c,b} &= (p_b{}^{\oplus} \to r_{c,b}) \wedge (p_b{}^{\ominus} \to \neg r_{c,b})\\
\psi'_{c,b} &= ((\neg p_b{}^{\oplus} \wedge \neg p_b{}^{\ominus}) \to (\omega_{c,b} \wedge \omega'_{c,b} \wedge \omega''_{c,b} \wedge \omega'''_{c,b}))\\
\omega_{c,b} &= ((p_c{}^{\oplus} \wedge o_{(b,c)}) \to \neg r_{c,b})\\
\omega'_{c,b} &= ((p_c{}^{\oplus} \wedge \neg o_{(b,c)} \wedge a_{(b,c)}) \to r_{c,b})\\
\omega''_{c,b} &= ((p_c{}^{\ominus} \wedge o_{(b,c)}) \to r_{c,b})\\
\omega'''_{c,b} &= ((p_c{}^{\ominus} \wedge \neg o_{(b,c)} \wedge a_{(b,c)}) \to \neg r_{c,b})\\
\psi_{c,c} &= (p_c{}^{\oplus} \to r_{c,c}) \wedge (p_c{}^{\ominus} \to \neg r_{c,c})\\
\psi'_{c,c} &= ((\neg p_c{}^{\oplus} \wedge \neg p_c{}^{\ominus}) \to (\omega_{c,c} \wedge \omega'_{c,c} \wedge \omega''_{c,c} \wedge \omega'''_{c,c}))\\
\omega_{c,c} &= ((p_c{}^{\oplus} \wedge o_{(c,c)}) \to \neg r_{c,c})\\
\omega'_{c,c} &= ((p_c{}^{\oplus} \wedge \neg o_{(c,c)} \wedge a_{(c,c)}) \to r_{c,c})\\
\omega''_{c,c} &= ((p_c{}^{\ominus} \wedge o_{(c,c)}) \to r_{c,c})\\
\omega'''_{c,c} &= ((p_c{}^{\ominus} \wedge \neg o_{(c,c)} \wedge a_{(c,c)}) \to \neg r_{c,c})
\end{aligned}
$$

$$
\begin{aligned}
evalCanon[D, P_S^3, R_{<S,par_D(S))>}] &= \gamma_1 \wedge \gamma_2 \wedge \gamma_3\\
\gamma_1 &= (p_a{}^{\oplus} \to (r_{a,b} \vee \neg r_{a,b})) \wedge (p_a{}^{\ominus} \to \neg(r_{a,b} \vee \neg r_{a,b}))\\
\gamma_2 &= (p_b{}^{\oplus} \to r_{b,b}) \wedge (p_b{}^{\ominus} \to \neg r_{b,b}) \wedge\\
\gamma_3 &= (p_c{}^{\oplus} \to (r_{c,c} \to r_{c,b})) \wedge (p_c{}^{\ominus} \to \neg(r_{c,c} \to r_{c,b}))
\end{aligned}
$$

Comparing the previous encoding with that from Example 5, we see that the link sensitive encoding seems quite more complex than the non link sensitive encoding. The main reason for this is the module encoding the link information which in matrix form introduces $\mathcal{O}(|S| * |L_D^?|)$ variables. In the worst case, this means that a number of variables that is cubic in the number of statements of the ADF may be required. Also, the module canon$[., ., ., ., .]$ is more complex than the corresponding module $\leq_i [., .]$ in the encoding in Example 5, requiring up to a quadratic number of variables (in the number of the *statements of the ADF) to encode the canonical models. The module* $\leq_i [.,.]$ *only needs a linear number of variables in the number of statements.*

On the other hand, as can be seen from the current example, if the number of unknown links is low the complexity of the module $links^M[.,.,.,.,.,.,.]$ *is reasonable. Also, and this is what has the most potential for boosting the performance of a link sensitive based approach compared to the non link sensitive approach, given a guessed interpretation* $v^?$ *for the statements* S *of the ADF* D *(and, hence, for the variables in* P_S^3*), as well as information about the links, the truth values of many of the variables encoding the canonical models should be easy to derive (depending on the availability of information about the link types as well as the level of refinement of the guessed interpretation* $v^?$*). In particular, it is to be expected that these values should be easy to derive for a QSAT solver. This is in contrast to the encoding in Example* 5 *where in principle all possible assignments to the variables in* P_S *must be considered when evaluating the module* $eval[.,.,.]$*, even when truth assignments to the variables in* P_S^3 *are given via a guess* $v^?$ *(except, that is, for those statements for which* $v^? = x$ *for* $x \in \{\boldsymbol{t}, \boldsymbol{f}\}$*).*

Concretely, for instance given the (unlucky) guess for an admissible interpretation $v^? = \{a \mapsto \boldsymbol{f}, b \mapsto \boldsymbol{u}, c \mapsto \boldsymbol{f}\}$*, and the available information about the links (represented by the function* $\iota_D(L_D)$*) in our example we have that*

$$
\begin{aligned}
Cred_{adm}(D,a) \equiv\ & \exists\ V\ \forall\ V'(\psi \wedge \psi' \wedge p_a{}^{\oplus})\Xi \\
\Xi :=\ & \{\{[A_{L_D^!} \cup O_{L_D^!}/\iota_D(L_D)][P_S^3/v^?(P_S^3)]\}\}^{R_{<S,par_D(S))>}} \\
V :=\ & \{a_{(b,c)}, o_{(b,c)}\}P_{<L_D^?,\exists,1>}P'_{<L_D^?,\exists,2>} \\
V' :=\ & P_{<L_D^?,\forall,1>}P'_{<L_D^?,\forall,2>}\{r_{b,b}, r_{c,b}\} \\
\psi :=\ & coh[P_S^3] \wedge links^M[D, A_{L_D}, O_{L_D}, P_{<L_D^?,\exists,1>}, P'_{<L_D^?,\exists,2>}, P_{<L_D^?,\forall,1>}, P'_{<L_D^?,\forall,2>}] \\
\psi' :=\ & (canon[D, A_{L_D}, O_{L_D}, P_S^3, R_{<S,par_D(S))>}] \rightarrow evalCanon[D, P_S^3, R_{<S,par_D(S))>}])
\end{aligned}
$$

(note the simplified versions of V *and* V'*), with the application of* $\Xi = \{\{[A_{L_D^!} \cup O_{L_D^!}/\iota_D(L_D)][P_S^3/v^?(P_S^3)]\}\}^{R_{<S,par_D(S))>}}$ *on the modules giving us:*

$$
\begin{aligned}
coh[P_S^3]\Xi =& \neg(\bot \wedge \top) \wedge \neg(\bot \wedge \bot) \wedge \neg(\bot \wedge \top) \\
links^M[D, A_{L_D}, O_{L_D}, P_{<L_D^?,\exists,1>}, P'{}_{<L_D^?,\exists,2>}, P_{<L_D^?,\forall,1>}, P'{}_{<L_D^?,\forall,2>}]\Xi =&\ \chi_1\Xi \wedge \chi_2 \\
\chi_1\Xi =& (\top \wedge \top \wedge \top \wedge \neg\bot \wedge \neg\bot \wedge \top) \\
\chi_2 =& \chi_2' \wedge \chi_2'' \wedge \chi_2''' \wedge \chi_2'''' \\
\chi_2' =& (o_{(b,c)} \rightarrow (\tau_1 \rightarrow \tau_2)) \\
\chi_2'' =& ((\tau_3 \rightarrow \tau_4) \rightarrow o_{(b,c)})
\end{aligned}
$$

$$\begin{aligned}
\chi_2''' &= (a_{(b,c)} \to (\tau_5 \to \tau_6))\\
\chi_2'''' &= ((\tau_7 \to \tau_8) \to a_{(b,c)})\\
\tau_1 &= ((p_{c,(b,c),\forall,O} \to p_{b,(b,c),\forall,O}) \wedge {p'}_{b,(b,c),\forall,O})\\
\tau_2 &= (p_{c,(b,c),\forall,O} \to {p'}_{b,(b,c),\forall,O})\\
\tau_3 &= ((p_{c,(b,c),\exists,O} \to p_{b,(b,c),\exists,O}) \wedge {p'}_{b,(b,c),\exists,O})\\
\tau_4 &= (p_{c,(b,c),\exists,O} \to {p'}_{b,(b,c),\exists,O})\\
\tau_5 &= (\neg(p_{c,(b,c),\forall,A} \to p_{b,(b,c),\forall,A}) \wedge {p'}_{b,(b,c),\forall,A})\\
\tau_6 &= \neg(p_{c,(b,c),\forall,A} \to {p'}_{b,(b,c),\forall,A})\\
\tau_7 &= (\neg(p_{c,(b,c),\exists,A} \to p_{b,(b,c),\exists,A}) \wedge {p'}_{b,(b,c),\exists,A})\\
\tau_8 &= \neg(p_{c,(b,c),\exists,A} \to {p'}_{b,(b,c),\exists,A})
\end{aligned}$$

$$canon[D, A_{L_D}, O_{L_D}, P_S^3, R_{<S,par_D(S))>}]\Xi = \bigwedge_{s_1 \in S, s_2 \in par_D(s_1)} (\psi_{s_1,s_2}\Xi \wedge \psi'_{s_1,s_2}\Xi)$$

$$\begin{aligned}
\psi_{a,b}\Xi &= (\bot \to \top) \wedge (\bot \to \neg\top)\\
\psi'_{a,b}\Xi &= ((\neg\bot \wedge \neg\bot) \to (\omega_{a,b}\Xi \wedge \omega'_{a,b}\Xi \wedge \omega''_{a,b}\Xi \wedge \omega'''_{a,b}\Xi))\\
\omega_{a,b}\Xi &= ((\bot \wedge \top) \to \neg\top)\\
\omega'_{a,b}\Xi &= ((\bot \wedge \neg\top \wedge \top) \to \top)\\
\omega''_{a,b}\Xi &= ((\top \wedge \top) \to \top)\\
\omega'''_{a,b}\Xi &= ((\top \wedge \neg\top \wedge \top) \to \neg\top)\\
\psi_{b,b}\Xi &= (\bot \to r_{b,b}) \wedge (\bot \to \neg r_{b,b})\\
\psi'_{b,b}\Xi &= ((\neg\bot \wedge \neg\bot) \to (\omega_{b,b}\Xi \wedge \omega'_{b,b}\Xi \wedge \omega''_{b,b}\Xi \wedge \omega'''_{b,b}\Xi))\\
\omega_{b,b}\Xi &= ((\bot \wedge \top) \to \neg r_{b,b})\\
\omega'_{b,b}\Xi &= ((\bot \wedge \neg\top \wedge \bot) \to r_{b,b})\\
\omega''_{b,b}\Xi &= ((\bot \wedge \top) \to r_{b,b})\\
\omega'''_{b,b}\Xi &= ((\bot \wedge \neg\top \wedge \bot) \to \neg r_{b,b})\\
\psi_{c,b}\Xi &= (\bot \to r_{c,b}) \wedge (\bot \to \neg r_{c,b})\\
\psi'_{c,b}\Xi &= ((\neg\bot \wedge \neg\bot) \to (\omega_{c,b}\Xi \wedge \omega'_{c,b}\Xi \wedge \omega''_{c,b}\Xi \wedge \omega'''_{c,b}\Xi))\\
\omega_{c,b}\Xi &= ((\bot \wedge o_{(b,c)}) \to \neg r_{c,b})\\
\omega'_{c,b}\Xi &= ((\bot \wedge \neg o_{(b,c)} \wedge a_{(b,c)}) \to r_{c,b})\\
\omega''_{c,b}\Xi &= ((\top \wedge o_{(b,c)}) \to r_{c,b})\\
\omega'''_{c,b}\Xi &= ((\top \wedge \neg o_{(b,c)} \wedge a_{(b,c)}) \to \neg r_{c,b})\\
\psi_{c,c}\Xi &= (\bot \to \bot) \wedge (\top \to \neg\bot)\\
\psi'_{c,c}\Xi &= ((\neg\bot \wedge \neg\top) \to (\omega_{c,c}\Xi \wedge \omega'_{c,c}\Xi \wedge \omega''_{c,c}\Xi \wedge \omega'''_{c,c}\Xi))\\
\omega_{c,c}\Xi &= ((\bot \wedge \bot) \to \neg\bot)\\
\omega'_{c,c}\Xi &= ((\bot \wedge \neg\bot \wedge \top) \to \bot)
\end{aligned}$$

$$\begin{aligned}
\omega''_{c,c}\Xi &= ((\top \wedge \bot) \rightarrow \bot) \\
\omega'''_{c,c}\Xi &= ((\top \wedge \neg\bot \wedge \top) \rightarrow \neg\bot) \\
evalCanon[D, P^3_S, R_{<S,par_D(S))>}]\Xi &= \ \gamma_1\Xi \wedge \gamma_2\Xi \wedge \gamma_3\Xi \\
\gamma_1\Xi &= (\bot \rightarrow (\top \vee \neg\top)) \wedge (\top \rightarrow \neg(\top \vee \neg\top)) \\
\gamma_2\Xi &= (\bot \rightarrow r_{b,b}) \wedge (\bot \rightarrow \neg r_{b,b})\wedge \\
\gamma_3\Xi &= (\bot \rightarrow (\bot \rightarrow r_{c,b})) \wedge (\top \rightarrow \neg(\bot \rightarrow r_{c,b}))
\end{aligned}$$

In other words, the semantic evaluation of the encoding can be simplified to the same degree that solving credulous acceptance for the ADF may be easier because of what is known about the link types of the ADF (as explained in Section 3.1.3.1). To repeat, the motivation behind the link information sensitive encodings is that QSAT solvers are also able to latch on to the extra information available in the link information sensitive encodings, thus boosting performance of an implementation of our encodings.

We will later on need to refer to our encoding for the admissible semantics, yet without the module specifying the link types. We thus conclude this section by giving its definition (for an ADF $D = (S, \{\phi_s\}_{s\in S})$):

$$\begin{aligned}
&\mathcal{E}'^{\#}_{adm}[D, P^3_S, A_{L_D}, O_{L_D}, R_{<S,par_D(S))>}] := \ coh[P^3_S] \wedge \forall R_{<S,par_D(S))>}\psi \\
&\psi := (canon[D, A_{L_D}, O_{L_D}, P^3_S, R_{<S,par_D(S))>}] \rightarrow evalCanon[D, P^3_S, R_{<S,par_D(S))>}]).
\end{aligned}$$

Note the new notation $\mathcal{E}'^{\#}_{adm}$ vs. the notation $\mathcal{E}'_{adm}$ used for the encoding that includes the module for the link types.

3.1.3.4 Encodings for the complete semantics

We turn now to the encoding of the complete semantics. For purposes of comparison, we again start by recalling the encoding from [Dil14]. Here, once more, $D = (S, C = \{\phi_s\}_{s\in S})$ is an ADF.

$$\begin{aligned}
\mathcal{E}_{com}[D, P^3_S, P_S, P'_S, P''_S] &:= \mathcal{E}_{adm}[D, P^3_S, P_S] \wedge \bigwedge_{s\in S}((\neg {p_s}^{\oplus} \wedge \neg {p_s}^{\ominus}) \rightarrow \psi_s) \\
\psi_s &:= \exists P'_S \cup P''_S(\leq_i [P^3_S, P'_S] \wedge \leq_i [P^3_S, P''_S] \wedge {\phi_s}^{P'_S} \wedge \neg{\phi_s}^{P''_S})
\end{aligned}$$

The first conjunct of $\mathcal{E}_{com}[D, ., ., ., .]$ ensures that any model of $\mathcal{E}_{com}[D, ., ., ., .]$ corresponds to an admissible interpretation v of D. The conjunct on the right is slightly more involved and expresses that v is also complete if $v(s) = \mathbf{u}$ for some $s \in S$ only if there exist $w, w' \in [v]_2$ such that $w(\phi_s) = \mathbf{t}$ and $w'(\phi_s) = \mathbf{f}$ (see the proof sketch of Proposition 6).

The link information sensitive encoding for the complete semantics is structurally similar to the non link information sensitive encoding, the difference now being that first of all we make use of the link information sensitive encoding of the admissible semantics $\mathcal{E}'_{adm}[.,.,.,.,.,.,.]$ from Section 3.1.3.3. Also, restating the properties used in the encoding $\mathcal{E}_{com}[.,.,.,.,.]$ in accordance with Corollary 2, we encode the fact that an admissible interpretation v is also complete if $v(s) = \mathbf{u}$ for some $s \in S$ only if there exists a $w \in [c^{\mathbf{f}}_{D,v,s}]_2$ such that $w(\phi_s) = \mathbf{t}$ as well as a $w' \in [c^{\mathbf{t}}_{D,v,s}]_2$ such that $w'(\phi_s) = \mathbf{f}$ ($c^{\mathbf{f}}_{D,v,s}$ and $c^{\mathbf{t}}_{D,v,s}$ being the canonical interpretations as defined in Proposition 4).

To encode the latter we now need a separate module for encoding (via the variables $R_{<s_1,par_D(s_1))>}$), first of all, the completions of the canonical interpretation $c^{\mathbf{t}}_{D,v,s_1}$ for an ADF D, interpretation v, and a statement s_1 of the ADF:

$$canon_{\mathbf{t}}[D,A_{L_D},O_{L_D},P^3_S,R_{<s_1,par_D(s_1))>}] := \bigwedge_{s_2 \in par_D(s_1)} (\psi_{s_2} \wedge \psi'_{s_2} \wedge \psi''_{s_2})$$
$$\psi_{s_2} := ({p_{s_2}}^{\oplus} \rightarrow r_{s_1,s_2})$$
$$\psi'_{s_2} := ({p_{s_2}}^{\ominus} \rightarrow \neg r_{s_1,s_2})$$
$$\psi''_{s_2} := ((\neg {p_{s_2}}^{\oplus} \wedge \neg {p_{s_2}}^{\ominus}) \rightarrow \tau_{s_2})$$
$$\tau_{s_2} := ((o_{(s_2,s_1)} \rightarrow \neg r_{s_1,s_2}) \wedge ((\neg o_{(s_2,s_1)} \wedge a_{(s_2,s_1)}) \rightarrow r_{s_1,s_2}))$$

Secondly, the following module encodes the completions of the canonical interpretation $c^{\mathbf{f}}_{D,v,s_1}$:

$$canon_{\mathbf{f}}[D,A_{L_D},O_{L_D},P^3_S,R_{<s_1,par_D(s_1))>}] := \bigwedge_{s_2 \in par_D(s_1)} (\psi_{s_2} \wedge \psi'_{s_2} \wedge \psi''_{s_2})$$
$$\psi_{s_2} := ({p_{s_2}}^{\oplus} \rightarrow r_{s_1,s_2})$$
$$\psi'_{s_2} := ({p_{s_2}}^{\ominus} \rightarrow \neg r_{s_1,s_2})$$
$$\psi''_{s_2} := ((\neg {p_{s_2}}^{\oplus} \wedge \neg {p_{s_2}}^{\ominus}) \rightarrow \tau_{s_2})$$
$$\tau_{s_2} := ((o_{(s_2,s_1)} \rightarrow r_{s_1,s_2}) \wedge ((\neg o_{(s_2,s_1)} \wedge a_{(s_2,s_1)}) \rightarrow \neg r_{s_1,s_2})).$$

The following is our link information sensitive encoding for the complete semantics. This is expressed formally in Proposition 6.

$$\mathcal{E}'_{com}[D,P^3_S,A_{L_D},O_{L_D},P_S,P'_S,R_{<S,par_D(S))>},R'_{<S,par_D(S))>},R''_{<S,par_D(S))>}] := \psi$$
$$\psi := \mathcal{E}'_{adm}[D,P^3_S,A_{L_D},O_{L_D},P_S,P'_S,R_{<S,par_D(S))>}] \wedge \bigwedge_{s \in S} ((\neg {p_s}^{\oplus} \wedge \neg {p_s}^{\ominus}) \rightarrow \psi_s)$$
$$\psi_s := \exists R'_{<s,par_D(s))>} \cup R''_{<s,par_D(s))>} (\tau_s \wedge \tau'_s)$$

$$\tau_s := canon_{\mathbf{t}}[D, A_{L_D}, O_{L_D}, P^3_S, R'_{<s,par_D(s))>}] \wedge \neg {\phi_s}^{R'_{<s,par_D(s))>}}$$
$$\tau'_s := canon_{\mathbf{f}}[D, A_{L_D}, O_{L_D}, P^3_S, R''_{<s,par_D(s))>}] \wedge {\phi_s}^{R''_{<s,par_D(s))>}}$$

Proposition 6. *Given an ADF $D = (S, \{\phi_s\}_{s\in S})$ and the list of indexed variables $V = P^3_S$, A_{L_D}, O_{L_D}, P_S, P'_S, $R_{<S,par_D(S))>}$, $R'_{<S,par_D(S))>}$, $R''_{<S,par_D(S))>}$, let $\mathcal{E}'_{com}$ be the function returning the QBF $\mathcal{E}'_{com}[D, V]$ having free variables in P^3_S. Then $\mathcal{E}'_{com}$ is a defining encoding function for the complete semantics.*

Proof. (sketch) Note first that for a three valued valuation v and an ADF $D = (S, C = \{\phi_s\}_{s\in S})$, $v \in com(D)$ if and only if (i) $v \in adm(D)$ and (ii) $v(s) = \mathbf{u}$ is the case for some $s \in S$ only if there exist $w, w' \in [v]_2$ such that $w(\phi_s) = \mathbf{t}$ and $w'(\phi_s) = \mathbf{f}$. That $v \in com(D)$ implies that v satisfies (i) and (ii) is immediate from the definition of complete interpretations (Definition 8). For the converse, (i), i.e. $v \leq_i \Gamma_D(v)$, implies that $v(s) = \Gamma_D(v)(s)$ for $s \in S$ such that $v(s) = \mathbf{t}$ and $v(s) = \mathbf{f}$. On the other hand (ii), which spells out the conditions under which $\Gamma_D(v)(s) = \mathbf{u}$ for some $s \in S$, implies that also $v(s) = \Gamma_D(v)(s)$ for $s \in S$ such that $v(s) = \mathbf{u}$.

Restating property (ii) in accordance with Corollary 2 we have that $v \in com(D)$ if and only if (i) $v \in adm(D)$ and (ii)' $v(s) = \mathbf{u}$ is the case for some $s \in S$ only if there exist $w \in [c^{\mathbf{f}}_{D,v,s}]_2$ such that $w(\phi_s) = \mathbf{t}$ as well as a $w' \in [c^{\mathbf{t}}_{D,v,s}]_2$ such that $w'(\phi_s) = \mathbf{f}$.

Coherence follows directly from Proposition 4 and the fact that for any two valued interpretation $\hat{v}$ such that $\hat{v} \models \mathcal{E}'_{com}[V]$ for the list of parameters $V = D$, P^3_S, A_{L_D}, O_{L_D}, P_S, P'_S, $R_{<S,par_D(S))>}$, $R'_{<S,par_D(S))>}$, $R''_{<S,par_D(S))>}$ it is also the case that $\hat{v} \models coh[P^3_S]$.

Proof of soundness and completeness involves showing that for a two valued interpretation $\hat{v}$, $\hat{v} \models \mathcal{E}'_{com}[V]$ if and only if the three valued interpretation v such that $\hat{v} \cong_{P^3_S} v$ satisfies (i) as defined above which is encoded via $\mathcal{E}'_{adm}[D, P^3_S, A_{L_D}, O_{L_D}, P_S, P'_S, R_{<S,par_D(S))>}]$ and (ii)' which is encoded via $\bigwedge_{s\in S}((\neg {p_s}^{\oplus} \wedge \neg {p_s}^{\ominus}) \to \psi_s)$ (where ψ_s is defined as in the definition of $\mathcal{E}'_{com}$). This is a technical but straightforward proof via lemmas 1 and 3, as well as Proposition 5. □

Turning to the encodings of the reasoning tasks, the following is the encoding of the verification problem in prenex normal form based on our link information sensitive defining encoding function $\mathcal{E}'_{com}$. We here assume that the statements of the ADF D are numbered, i.e. $S = \{s_1, \ldots, s_n\}$

$$Ver_{com}(D, v) \equiv \ (\exists\ V\ \forall\ V'(\psi \wedge \psi') \wedge \exists\ V''\ X)[P^3_S/v(S)]$$
$$V := A_{L_D} O_{L_D} P_{<L^?_D,\exists,1>} P'_{<L^?_D,\exists,2>}$$
$$V' := P_{<L^?_D,\forall,1>} P'_{<L^?_D,\forall,2>} R_{<S,par_D(S))>}$$

$$
\begin{aligned}
V'' &:= R^{s'_1}{}_{<s_1, par_D(s_1))>}\ R^{s''_1}{}_{<s_1, par_D(s_1))>} \dots R^{s'_n}{}_{<s_n, par_D(s_n))>}\ R^{s''_n}{}_{<s_n, par_D(s_n))>}\\
\psi &:= coh[P^3_S] \land links^M[D, A_{L_D}, O_{L_D}, P_{<L^?_D,\exists,1>}, P'_{<L^?_D,\exists,2>}, P_{<L^?_D,\forall,1>}, P'_{<L^?_D,\forall,2>}]\\
\psi' &:= (canon[D, A_{L_D}, O_{L_D}, P^3_S, R_{<S,par_D(S))>}] \rightarrow evalCanon[D, P^3_S, R_{<S,par_D(S))>}])\\
X &:= \bigwedge_{1\le i\le n} ((\neg p_{s_i}{}^{\oplus} \land \neg p_{s_i}{}^{\ominus}) \rightarrow (\chi_{s_i} \land \chi'_{s_i}))\\
\chi_{s_i} &:= canon_{\mathbf{t}}[D, A_{L_D}, O_{L_D}, P^3_S, R^{s'_i}{}_{<s_i,par_D(s_i))>}] \land \neg\phi_{s_i}{}^{R^{s'_i}{}_{<s_i,par_D(s_i))>}}\ (1 \le i \le n)\\
\chi'_{s_i} &:= canon_{\mathbf{f}}[D, A_{L_D}, O_{L_D}, P^3_S, R^{s''_i}{}_{<s_i,par_D(s_i))>}] \land \phi_{s_i}{}^{R^{s''_i}{}_{<s_i,par_D(s_i))>}}\ (1 \le i \le n).
\end{aligned}
$$

When $L^?_D = \emptyset$ the encoding reduces to

$$
\begin{aligned}
&Ver_{com}(D, v) \equiv\ (\forall\ V'(\psi \land \psi') \land \exists\ V''\ X)[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\\
&\quad V' := R_{<S,par_D(S))>}\\
&\quad V'' := R^{s'_1}{}_{<s_1, par_D(s_1))>}\ R^{s''_1}{}_{<s_1, par_D(s_1))>} \dots R^{s'_n}{}_{<s_n, par_D(s_n))>}\ R^{s''_n}{}_{<s_n, par_D(s_n))>},
\end{aligned}
$$

which matches the complexity of verification for the complete semantics (DP- complete) in the sense that the encoding mirrors the structure of the SAT-UNSAT problem.

Moreover, when the ADF D is a BADF the encoding can be further reduced to (here we implicitly extend $\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\}\}^{R_{<S,par_D(S))>}}$ to the variables in $R^{s'_1}{}_{<s_1,par_D(s_1))>} \cup \ldots \cup R^{s''_n}{}_{<s_n,par_D(s_n))>}$[5]):

$$
\begin{aligned}
&Ver_{com}(D, v) \equiv\ (\Psi \land\ X')\\
&\quad \Psi := (\psi \land \psi')\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\}\}^{R_{<S,par_D(S))>}}\\
&\quad X' := X\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\}\}^{V''}\\
&\quad V'' := R^{s'_1}{}_{<s_1, par_D(s_1))>}\ R^{s''_1}{}_{<s_1, par_D(s_1))>} \dots R^{s'_n}{}_{<s_n, par_D(s_n))>}\ R^{s''_n}{}_{<s_n, par_D(s_n))>}
\end{aligned}
$$

i.e., the encoding returns a propositional formula with true constants only. This matches the complexity of the verification problem for BADFs which is in P.

For credulous reasoning w.r.t. the complete semantics, we obtain link information sensitive encodings via the encoding for the admissible semantics (Section 3.1.3.3) and

[5]Formally, $\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\}\}^{R^{s'_1}{}_{<s_1,par_D(s_1))>} \cdots R^{s''_n}{}_{<s_n,par_D(s_n))>}} := \{\{\{\{\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\}\}^{R^{s'_1}{}_{<s_1,par_D(s_1))>}}\}\}^{\cdots}\}\}^{R^{s''_n}{}_{<s_n,par_D(s_n))>}}$.

the fact that $Cred_{com}(D, s^*) = Cred_{adm}(D, s^*)$ for any ADFs D and statement s^*. This follows from the fact that any complete interpretation is admissible and any admissible interpretation v for which $v(s^*) = \mathbf{t}$ can be extended to a complete interpretation v' for which $v'(s^*) = \mathbf{t}$.

Regarding skeptical reasoning, we obtain encodings via $Skept_{com}(D, s^*) = Skept_{grd}(D, s^*)$ and the encoding for skeptical reasoning for the grounded semantics we give in Section 3.1.3.6. The reason for $Skept_{com}(D, s^*) = Skept_{grd}(D, s^*)$ is that the grounded interpretation is the minimally informative (w.r.t. $\leq_i$) complete interpretation.

As we will need it later on, we conclude this section by giving the defining encoding of the complete semantics without making use of the module for encoding the link types. This version simply makes use of the encoding of the admissible semantics without the links module defined in Section 3.1.3.3 ($\mathcal{E}'^{\#}_{adm}$):

$$\begin{aligned}
\mathcal{E}'^{\#}_{com}[D, P^3_S, A_{L_D}, O_{L_D}, P_S, P'_S, R_{<S,par_D(S))>}, R'_{<S,par_D(S))>}, R''_{<S,par_D(S))>}] &:= \psi \\
\psi &:= \mathcal{E}'^{\#}_{adm}[D, P^3_S, A_{L_D}, O_{L_D}, P_S, P'_S, R_{<S,par_D(S))>}] \wedge \bigwedge_{s \in S} ((\neg {p_s}^{\oplus} \wedge \neg {p_s}^{\ominus}) \rightarrow \psi_s) \\
\psi_s &:= \exists R'_{<s,par_D(s))>} \cup R''_{<s,par_D(s))>} (\tau_s \wedge \tau'_s) \\
\tau_s &:= canon_{\mathbf{t}}[D, A_{L_D}, O_{L_D}, P^3_S, R'_{<s,par_D(s))>}] \wedge \neg {\phi_s}^{R'_{<s,par_D(s))>}} \\
\tau'_s &:= canon_{\mathbf{f}}[D, A_{L_D}, O_{L_D}, P^3_S, R''_{<s,par_D(s))>}] \wedge {\phi_s}^{R''_{<s,par_D(s))>}}.
\end{aligned}$$

3.1.3.5 Encodings for the preferred semantics

As for the previous sections on link information encodings, we start this section about encodings for the preferred semantics by recalling the defining encoding function from [Dil14]:

$$\begin{aligned}
\mathcal{E}_{prf}[D, P^3_S, P_S, P'^3_S, P'_S] &:= \psi \wedge \psi' \\
\psi &:= \mathcal{E}_{adm}[D, P^3_S, P_S] \\
\psi' &:= \forall P'^3_S(\mathcal{E}_{adm}[D, P'^3_S, P'_S] \rightarrow (\leq_i [P^3_S, P'^3_S] \rightarrow \leq_i [P'^3_S, P^3_S]))
\end{aligned}$$

The defining encoding function returns a QBF that specifies that the result should correspond to an admissible interpretation v and for any admissible interpretation v' with greater or equal information content it must be the case that $v(s) = v'(s)$ for all $s \in S$, i.e. v is an admissible interpretation that is maximally informative with respect to $\leq_i$. Thus, the encoding pretty much reflects the definition of the preferred semantics (Definition 8).

The only difference between the link information sensitive defining encoding function for the preferred semantics and the non link information sensitive encoding is that, as is to be expected, for the link information sensitive encoding we use the link information sensitive version of the defining encoding function for the admissible semantics (Section 3.1.3.3):

$$\begin{aligned}
&\mathcal{E}'_{prf}[D, P^3_S, A_{L_D}, O_{L_D}, P_S, P'_S, R_{<S,par_D(S))>}, P'^3_S, R'_{<S,par_D(S))>}] := \ \psi \wedge \psi' \\
&\psi := \mathcal{E}'_{adm}[D, P^3_S, A_{L_D}, O_{L_D}, P_S, P'_S, R_{<S,par_D(S))>}] \\
&\psi' := \forall P'^3_S(\mathcal{E}'^{\#}_{adm}[D, P'^3_S, A_{L_D}, O_{L_D}, R'_{<S,par_D(S))>}] \rightarrow (\leq_i [P^3_S, P'^3_S] \rightarrow \leq_i [P'^3_S, P^3_S]))
\end{aligned}$$

Note that the first use of defining encoding function for the admissible semantics within our link information sensitive encoding for the preferred semantics is the version with the module for the link types, while the second use is without the module $links[., ., ., ., .]$. Proposition 7 more formally expresses the fact that $\mathcal{E}'_{prf}$ is a defining encoding function for the preferred semantics.

Proposition 7. *Given an ADF $D = (S, \{\phi_s\}_{s \in S})$ and the list of indexed variables $V = P^3_S$, A_{L_D}, O_{L_D}, P_S, P'_S, $R_{<S,par_D(S))>}$, P'^3_S, $R'_{<S,par_D(S))>}$, let $\mathcal{E}'_{prf}$ be the function returning the QBF with free variables in P^3_S, $\mathcal{E}'_{prf}[D, V]$. Then $\mathcal{E}'_{prf}$ is a defining encoding function for the preferred semantics.*

Proof. (sketch) The proof is almost identical to the proof about the defining encoding function $\mathcal{E}_{prf}$ in [DWW15] (Proposition 3.14; in turn, a restatement of Lemma 3.5.2 in [Dil14]). The main difference is in the use of Proposition 5 (together with lemmas 1, 3, and 4). Also, the equivalence of using $\mathcal{E}'^{\#}_{adm}$ and (a second application of) $\mathcal{E}'_{adm}$ in the second use of the module for the admissible semantics within the definition of $\mathcal{E}'_{prf}$ should be easy to convince oneself of. □

Moving on to the encodings of the reasoning tasks of interest to this work, the following is the encoding of the verification problem for the preferred semantics in prenex normal form:

$$\begin{aligned}
&Ver_{prf}(D, v) \equiv \ \exists V \forall V' \exists V''(\psi \wedge (\chi \rightarrow \omega))[P^3_S/v(S)] \\
&\quad V := A_{L_D} O_{L_D} P_{<L^?_D,\exists,1>} P'_{<L^?_D,\exists,2>} \\
&\quad V' := P_{<L^?_D,\forall,1>} P'_{<L^?_D,\forall,2>} R_{<S,par_D(S))>} P'^3_S \\
&\quad V'' := R'_{<S,par_D(S))>} \\
&\quad \psi := \psi_1 \wedge \psi_2 \\
&\quad \psi_1 := coh[P^3_S] \wedge links^M[D, A_{L_D}, O_{L_D}, P_{<L^?_D,\exists,1>}, P'_{<L^?_D,\exists,2>}, P_{<L^?_D,\forall,1>}, P'_{<L^?_D,\forall,2>}]
\end{aligned}$$

$$\begin{aligned}\psi_2 &:= (canon[D, A_{L_D}, O_{L_D}, P^3_S, R_{<S,par_D(S))>}] \rightarrow evalCanon[D, P^3_S, R_{<S,par_D(S))>}])\\ \chi &:= coh[P'^3_S] \wedge \chi'\\ \chi' &:= (canon[D, A_{L_D}, O_{L_D}, P'^3_S, R'_{<S,par_D(S))>}] \rightarrow evalCanon[D, P'^3_S, R'_{<S,par_D(S))>}])\\ \omega &:= (\leq_i [P^3_S, P'^3_S] \rightarrow \leq_i [P'^3_S, P^3_S]))\end{aligned}$$

If the types of all links are known, the encoding reduces to the following:

$$\begin{aligned}Ver_{prf}(D, v) &\equiv\ \forall V' \exists V''(\psi \wedge (\chi \rightarrow \omega))[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\\ V' &:= R_{<S,par_D(S))>} P'^3_S\\ V'' &:= R'_{<S,par_D(S))>}\end{aligned}$$

i.e. a Π_2 QBF, thus matching the complexity of the verification problem for the preferred semantics. (Π^P_2-complete). Moreover, if the ADF is a BADF the encoding can be further simplified to return a Π_1 QBF, matching the complexity of the verification problem for BADFs (coNP-complete):

$$\begin{aligned}Ver_{prf}(D, v) &\equiv\ \forall P'^3_S(\psi \wedge (\chi \rightarrow \omega))\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\}\}^{V''''}\\ V'''' &:= R_{<S,par_D(S))>} \cup R'_{<S,par_D(S))>}\end{aligned}$$

Here again we extend the notation $\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\}\}^{R_{<S,par_D(S))>}}$ to also propagate the information given by the replacement $[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]$ to the variables $R'_{<S,par_D(S))>}$ (in addition to the variables $R_{<S,par_D(S))>}$) in the obvious manner (see Footnote 5).

Skeptical reasoning for the preferred semantics can be encoded in the following manner:

$$\begin{aligned}Skept_{prf}(D, s^*) &\equiv\ \forall V \exists V' \forall V''((\psi \wedge (\chi \rightarrow \omega)) \rightarrow s^{*\oplus})\\ V &:= P^3_S A_{L_D} O_{L_D} P_{<L^?_D,\exists,1>} P'_{<L^?_D,\exists,2>}\\ V' &:= P_{<L^?_D,\forall,1>} P'_{<L^?_D,\forall,2>} R_{<S,par_D(S))>} P'^3_S\\ V'' &:= R'_{<S,par_D(S))>}\\ \psi &:= \psi_1 \wedge \psi_2\\ \psi_1 &:= coh[P^3_S] \wedge links^M[D, A_{L_D}, O_{L_D}, P_{<L^?_D,\exists,1>}, P'_{<L^?_D,\exists,2>}, P_{<L^?_D,\forall,1>}, P'_{<L^?_D,\forall,2>}]\end{aligned}$$

$$\begin{aligned}
\psi_2 &:= \big(canon[D, A_{L_D}, O_{L_D}, P_S^3, R_{<S,par_D(S))>}] \rightarrow evalCanon[D, P_S^3, R_{<S,par_D(S))>}]\big)\\
\chi &:= coh[{P'}_S^3] \wedge \chi'\\
\chi' &:= \big(canon[D, A_{L_D}, O_{L_D}, {P'}_S^3, {R'}_{<S,par_D(S))>}] \rightarrow evalCanon[D, {P'}_S^3, {R'}_{<S,par_D(S))>}]\big)\\
\omega &:= (\leq_i [P_S^3, {P'}_S^3] \rightarrow \leq_i [{P'}_S^3, P_S^3])).
\end{aligned}$$

This is a Π_3 QBF thus matching the complexity of skeptical reasoning for the preferred semantics (Π_3^P-complete), but can be further simplified when $L_D^? = \emptyset$. In particular, when the ADF D is a BADF, the encoding reduces to the Π_2 QBF:

$$\begin{aligned}
Skept_{prf}(D, s^*) \equiv\ & \forall P_S^3 \exists {P'}_S^3((\psi \wedge (\chi \rightarrow \omega)) \rightarrow s^{*\oplus})\Xi\\
\Xi :=& \{\{[A_{L_D^!} \cup O_{L_D^!} / \iota_D(L_D)][P_S^3/v(S)]\}\}^{V''''}\\
V'''' :=\ & R_{<S,par_D(S))>} \cup {R'}_{<S,par_D(S))>}
\end{aligned}$$

This, once more, matches the complexity of skeptical reasoning for the preferred semantics for BADFs (Π_2^P-complete).

To conclude this subsection we remind the reader that $Cred_{prf}(D, s^*) \equiv Cred_{adm}(D, s^*)$ for any ADF D and statement s^*. The reason is that, on the one hand, any preferred interpretation is an admissible interpretation. On the other hand, the fact that preferred interpretations are the maximally informative admissible interpretations (w.r.t. $\leq_i$), implies that the existence of an admissible interpretation v s.t. $v(s^*) = \mathbf{t}$ means that there must be a preferred interpretation $v \leq_i v'$ and, thus, also $v'(s^*) = \mathbf{t}$. Therefore, the encoding for credulous reasoning w.r.t. the admissible semantics from Section 3.1.3.3 provides us with a link information sensitive encoding also for credulous reasoning for the preferred semantics.

3.1.3.6 Encodings for the grounded semantics

A straightforward defining encoding function for the grounded semantics results from making use of the encoding for the complete semantics and basically recasting the definition of the grounded semantics (Definition 8)-, i.e. the fact that the (unique) grounded interpretation is the minimally informative (w.r.t. $\leq_i$) complete interpretation,- in QBF terms. In the version of [Dil14] this encoding strategy results in the following defining encoding function:

$$\begin{aligned}
&\mathcal{E}'_{grd}[D, P_S^3, P_S, {P'}_S, {P''}_S, {P'}_S^3, {P'''}_S, {P''''}_S, {P'''''}_S] := \psi\\
&\psi := \mathcal{E}_{com}[D, P_S^3, P_S, {P'}_S, {P''}_S] \wedge \forall {P'}_S^3(\mathcal{E}_{com}[D, {P'}_S^3, {P'''}_S, {P''''}_S, {P'''''}_S] \rightarrow \leq_i [P_S^3, {P'}_S^3]).
\end{aligned}$$

A link information sensitive variant of this encoding results from making use of the link information sensitive version of the defining encoding function for the complete semantics:

$$\begin{aligned}
&\mathcal{E}''_{grd}[D, V^{\mathcal{L}}, V'^{\mathcal{L}}] := \psi\\
&V := P^3_S\ A_{L_D}\ O_{L_D}\ P'_S\ R_{<S,par_D(S))>}\ R'_{<S,par_D(S))>}\ R''_{<S,par_D(S))>}\\
&V' := P'^3_S\ A_{L_D}\ O_{L_D}\ P''_S\ R'''_{<S,par_D(S))>}\ R''''_{<S,par_D(S))>}\ R'''''_{<S,par_D(S))>}\\
&\psi := \mathcal{E}'_{com}[D, P^3_S, A_{L_D}, O_{L_D}, P'_S, R_{<S,par_D(S))>}, R'_{<S,par_D(S))>}, R''_{<S,par_D(S))>}] \wedge \psi'\\
&\psi' := \forall P'^3_S(\psi'' \to \leq_i [P^3_S, P'^3_S])\\
&\psi'' := \mathcal{E}'^{\#}_{com}[D, P'^3_S, A_{L_D}, O_{L_D}, P''_S, R'''_{<S,par_D(S))>}, R''''_{<S,par_D(S))>}, R'''''_{<S,par_D(S))>}].
\end{aligned}$$

Note that the second use of the encoding for the complete semantics ($\mathcal{E}'^{\#}_{com}$) in $\mathcal{E}''_{grd}$ is the version which does not include the module for specifying the link types of the ADF (see Section 3.1.3.4). We remind the reader also of the notation $V^{\mathcal{L}} := V_1, \ldots, V_m$ we introduced in Section 3.1.2 for turning a "block" of sets of variables $V = V_1 \ldots V_m$ into a (comma separated) "list" of sets of variables.

Although the encodings $\mathcal{E}'_{grd}$ and $\mathcal{E}''_{grd}$ are modular and directly reflect the definition of the grounded semantics, both lead to non complexity sensitive encodings of the reasoning tasks that are of interest to this work (when using the technique for making encodings of the reasoning tasks out of the defining encoding functions described in Section 3.1.1.1). For this reason, already in [Dil14] we gave an alternative complexity sensitive (yet, non link information sensitive) version of the encoding for the grounded semantics.

We have already described the latter alternative complexity sensitive defining encoding function for the grounded semantics $\mathcal{E}_{grd}$ in Section 3.1.2. We remind the reader that it is based on the fact that (see [SW15]) $v \in grd(D)$ for an interpretation v and an ADF $D = (S, \{\phi_s\}_{s\in S})$ if v is the $\leq_i$-minimal interpretation satisfying i) for each $s \in S$ such that $v(s) = \mathbf{t}$ there exists an interpretation $w \in [v]_2$ for which $w(\phi_s) = \mathbf{t}$, ii) for each $s \in S$ such that $v(s) = \mathbf{f}$ there exists an interpretation $w \in [v]_2$ for which $w(\phi_s) = \mathbf{f}$, and finally iii) for each $s \in S$ such that $v(s) = \mathbf{u}$ there exist interpretations $w_1 \in [v]_2$ and $w_2 \in [v]_2$ such that $w_1(\phi_s) = \mathbf{t}$ and $w_2(\phi_s) = \mathbf{f}$. We called an interpretation v for an ADF D that satisfies properties i)-iii) a candidate for being the grounded interpretation.

Using Corollary 2, we can restate items i)-iii) that candidates for the grounded interpretation need to satisfy as follows: i)' for each $s \in S$ such that $v(s) = \mathbf{t}$ there exists an interpretation $w \in [c^{\mathbf{f}}_{D,v,s}]_2$ for which $w(\phi_s) = \mathbf{t}$, ii) for each $s \in S$ such that $v(s) = \mathbf{f}$ there exists an interpretation $w \in [c^{\mathbf{t}}_{D,v,s}]_2$ for which $w(\phi_s) = \mathbf{f}$, and iii) for each $s \in S$ such that $v(s) = \mathbf{u}$ there exist interpretations $w_1 \in [c^{\mathbf{f}}_{D,v,s}]_2$ and $w_2 \in [c^{\mathbf{t}}_{D,v,s}]_2$ such that $w_1(\phi_s) = \mathbf{t}$ and $w_2(\phi_s) = \mathbf{f}$. The following link information sensitive defining encoding function for the grounded semantics makes direct use of the characterisation of the grounded interpretation as the minimally informative (w.r.t. $\leq_i$) of the candidates for being the grounded interpretation as expressed via properties i)'-iii)':

$$\mathcal{E}'''_{grd}[D, V'''^{\mathcal{L}}] := coh[P^3_S] \wedge \psi \wedge\ (\forall P'^3_S\ \omega)$$

$$
\begin{aligned}
V''' &:= P^3_S\ P'^3_S\ A_{L_D}\ O_{L_D}\ P_S\ P'_S\ R_{<S,par_D(S))>}\ R'_{<S,par_D(S))>}\\
\psi &:= links[D, A_{L_D}, O_{L_D}, P_S, P'_S] \wedge prop2[D, P^3_S, A_{L_D}, O_{L_D}, R_{<S,par_D(S))>}, R'_{<S,par_D(S))>}]\\
\omega &:= \big((coh[P'^3_S] \wedge prop2[D, P'^3_S, A_{L_D}, O_{L_D}, R_{<S,par_D(S))>}, R'_{<S,par_D(S))>}]\big) \rightarrow_{\leq_i} [P^3_S, P'^3_S]\big).
\end{aligned}
$$

Here, the module $prop2[.,.,.,.,.,.]$ is used to encode the properties i')-iii') and makes use of the modules $canon_{\mathbf{t}}[.,.,.,.,.]$ and $canon_{\mathbf{f}}[.,.,.,.,.]$ defined in Section 3.1.3.4:

$$
\begin{aligned}
&prop2[D, P^3_S, A_{L_D}, O_{L_D}, R_{<S,par_D(S))>}, R'_{<S,par_D(S))>}] := \bigwedge_{s \in S} (\psi_s \wedge \psi'_s \wedge \psi''_s)\\
&\quad \psi_s := ({p_s}^{\oplus} \rightarrow \exists R_{<s,par_D(s))>}(canon_{\mathbf{f}}[D, A_{L_D}, O_{L_D}, P^3_S, R_{<s,par_D(s))>}] \wedge {\phi_s}^{R_{<s,par_D(s))>}}))\\
&\quad \psi'_s := ({p_s}^{\ominus} \rightarrow \exists R_{<s,par_D(s))>}(canon_{\mathbf{t}}[D, A_{L_D}, O_{L_D}, P^3_S, R_{<s,par_D(s))>}] \wedge \neg{\phi_s}^{R_{<s,par_D(s))>}}))\\
&\quad \psi''_s := ((\neg {p_s}^{\oplus} \wedge \neg {p_s}^{\ominus}) \rightarrow \psi'''_s)\\
&\quad \psi'''_s := \exists R_{<s,par_D(s))>} \cup R'_{<s,par_D(s))>}(\psi''''_s \wedge {\phi_s}^{R_{<s,par_D(s))>}} \wedge \neg{\phi_s}^{R'_{<s,par_D(s))>}})\\
&\quad \psi''''_s := canon_{\mathbf{f}}[D, A_{L_D}, O_{L_D}, P^3_S, R_{<s,par_D(s))>}] \wedge canon_{\mathbf{t}}[D, A_{L_D}, O_{L_D}, P^3_S, R'_{<s,par_D(s))>}].
\end{aligned}
$$

Proposition 8 states that $\mathcal{E}''_{grd}$ as well as $\mathcal{E}'''_{grd}$ are indeed defining encoding functions for the grounded semantics.

Proposition 8. *Let $D = (S, \{\phi_s\}_{s\in S})$ be an ADF with $S = \{s_1, \ldots, s_n\}$. Also, let V be the list of sets of indexed variables P^3_S, A_{L_D}, O_{L_D}, P'_S, $R_{<S,par_D(S))>}$, $R'_{<S,par_D(S))>}$, $R''_{<S,par_D(S))>}$. Moreover, V' is the list of sets of indexed variables P'^3_S, A_{L_D}, O_{L_D}, P''_S, $R'''_{<S,par_D(S))>}$, $R''''_{<S,par_D(S))>}$, $R'''''_{<S,par_D(S))>}$, ${P^{s'_1}}_S$. Finally, V''' is the list of sets of indexed variables P^3_S, P'^3_S, A_{L_D}, O_{L_D}, P_S, P'_S, $R_{<S,par_D(S))>}$, $R'_{<S,par_D(S))>}$. Then both the functions $\mathcal{E}''_{grd}$ as well as $\mathcal{E}'''_{grd}$, returning the QBFs with free variables in P^3_S $\mathcal{E}''_{grd}[D, V, V']$ and $\mathcal{E}'''_{grd}[D, V''']$ respectively, are defining encoding functions for the grounded semantics.*

Proof. (sketch) The defining encoding function $\mathcal{E}''_{grd}$ reflects the definition of the grounded semantics (Definition 8), while $\mathcal{E}'''_{grd}$ uses the characterisation of the grounded semantics as the minimally informative (w.r.t. $\leq_i$) candidate for being the grounded interpretation ([SW15]). The latter being characterised via properties i)'-iii)' as detailed above. The proof for $\mathcal{E}''_{grd}$ requires the use of Proposition 6, while that for $\mathcal{E}'''_{grd}$ requires the use of lemmas 4 and 7. Both proofs carry through by additionally also making use of lemmas 1, 3, and 5. $\square$

For encoding the reasoning tasks for the grounded semantics in prenex normal form we need the matrix form of the module $prop2[D,.,.,.,.,.]$. Assuming the statements S of the ADF D are numbered -, i.e. $S = \{s_1, \ldots, s_n\}$,- the matrix form is defined as shown next:

$$
\begin{aligned}
prop2^M[D, P_S^3, A_{L_D}, O_{L_D}, {V_1}^{\mathcal{L}}, \ldots, {V_n}^{\mathcal{L}}] :=& \bigwedge_{1 \leq i \leq n} (\psi_{s_i} \land \psi'_{s_i} \land \psi''_{s_i}) \\
V_i :=& \ R^{s'_i}{}_{<s_i, par_D(s_i))>} \ R^{s''_i}{}_{<s_i, par_D(s_i))>} \ R^{s'''_i}{}_{<s_i, par_D(s_i))>} \ R^{s''''_i}{}_{<s_i, par_D(s_i))>} \ (1 \leq i \leq n) \\
\psi_{s_i} :=& ({p_{s_i}}^{\oplus} \rightarrow \chi_i) \ (1 \leq i \leq n) \\
\chi_i :=& (canon_{\mathbf{f}}[D, A_{L_D}, O_{L_D}, P_S^3, R^{s'_i}{}_{<s_i, par_D(s_i))>}] \land {\phi_{s_i}}^{R^{s'_i}{}_{<s_i, par_D(s_i))>}}) \ (1 \leq i \leq n) \\
\psi'_{s_i} :=& ({p_{s_i}}^{\ominus} \rightarrow \chi'_i) \ (1 \leq i \leq n) \\
\chi'_i :=& (canon_{\mathbf{t}}[D, A_{L_D}, O_{L_D}, P_S^3, R^{s''_i}{}_{<s_i, par_D(s_i))>}] \land \neg {\phi_{s_i}}^{R^{s''_i}{}_{<s_i, par_D(s_i))>}}) \ (1 \leq i \leq n) \\
\psi''_{s_i} :=& ((\neg {p_{s_i}}^{\oplus} \land \neg {p_{s_i}}^{\ominus}) \rightarrow \psi'''_{s_i}) \ (1 \leq i \leq n) \\
\psi'''_{s_i} :=& (\psi''''_{s_i} \land \psi'''''_{s_i} \land {\phi_{s_i}}^{R^{s'''_i}{}_{<s_i, par_D(s_i))>}} \land \neg {\phi_{s_i}}^{R^{s''''_i}{}_{<s_i, par_D(s_i))>}}) \ (1 \leq i \leq n) \\
\psi''''_{s_i} :=& canon_{\mathbf{f}}[D, A_{L_D}, O_{L_D}, P_S^3, R^{s'''_i}{}_{<s_i, par_D(s_i))>}] \ (1 \leq i \leq n) \\
\psi'''''_{s_i} :=& canon_{\mathbf{t}}[D, A_{L_D}, O_{L_D}, P_S^3, R^{s''''_i}{}_{<s_i, par_D(s_i))>}] \ (1 \leq i \leq n).
\end{aligned}
$$

The verification problem w.r.t. an ADF D, an interpretation v, and the grounded semantics can then be encoded as follows:

$$
\begin{aligned}
Ver_{grd}(D, v) \equiv& \ (\exists V \forall V''(\psi \land \psi')) \land (\forall V''' \chi)[P_S^3 / v(S)] \\
V :=& \ A_{L_D} O_{L_D} P_{<L_D^?, \exists, 1>} P'_{<L_D^?, \exists, 2>} \ V_1 \ldots V_n \\
V'' :=& \ P_{<L_D^?, \forall, 1>} P'_{<L_D^?, \forall, 2>} \\
V''' :=& \ {P'}_S^3 \ V'_1 \ldots V'_n \\
V_i :=& \ R^{s'_i}{}_{<s_i, par_D(s_i))>} \ R^{s''_i}{}_{<s_i, par_D(s_i))>} \ R^{s'''_i}{}_{<s_i, par_D(s_i))>} \ R^{s''''_i}{}_{<s_i, par_D(s_i))>} \ \ (1 \leq i \leq n) \\
V'_i :=& \ T^{s'_i}{}_{<s_i, par_D(s_i))>} \ T^{s''_i}{}_{<s_i, par_D(s_i))>} \ T^{s'''_i}{}_{<s_i, par_D(s_i))>} \ T^{s''''_i}{}_{<s_i, par_D(s_i))>} \ \ (1 \leq i \leq n) \\
\psi :=& \ coh[P_S^3] \land links^M[D, A_{L_D}, O_{L_D}, P_{<L_D^?, \exists, 1>}, P'_{<L_D^?, \exists, 2>}, P_{<L_D^?, \forall, 1>}, P'_{<L_D^?, \forall, 2>}] \\
\psi' :=& \ prop2^M[D, P_S^3, A_{L_D}, O_{L_D}, {V_1}^{\mathcal{L}}, \ldots, {V_n}^{\mathcal{L}}] \\
\chi :=& ((coh[{P'}_S^3] \land prop2^M[D, {P'}_S^3, A_{L_D}, O_{L_D}, {V'_1}^{\mathcal{L}}, \ldots, {V'_n}^{\mathcal{L}}]) \rightarrow \leq_i [P_S^3, {P'}_S^3]).
\end{aligned}
$$

When the types of all of the links of the ADF D are known, the encoding can be simplified to

$$
Ver_{grd}(D, v) \equiv \ (\exists V(\psi \land \psi')) \land (\forall V''' \chi)[A_{L_D^!} \cup O_{L_D^!} / \iota_D(L_D)][P_S^3 / v(S)]
$$

$$
\begin{aligned}
V &:= V_1 \ldots V_n \\
V''' &:= P'^3_S\ V'_1 \ldots V'_n \\
V_i &:= R^{s'_i}{}_{<s_i,par_D(s_i))>}\ R^{s''_i}{}_{<s_i,par_D(s_i))>}\ R^{s'''_i}{}_{<s_i,par_D(s_i))>}\ R^{s''''_i}{}_{<s_i,par_D(s_i))>} \quad (1 \leq i \leq n) \\
V'_i &:= T^{s'_i}{}_{<s_i,par_D(s_i))>}\ T^{s''_i}{}_{<s_i,par_D(s_i))>}\ T^{s'''_i}{}_{<s_i,par_D(s_i))>}\ T^{s''''_i}{}_{<s_i,par_D(s_i))>} \quad (1 \leq i \leq n)
\end{aligned}
$$

which reflects the complexity of the SAT-UNSAT problem and thus matches the complexity of the verification problem w.r.t. the grounded semantics which is DP-complete.

Given that the complexity of the verification problem for the grounded semantics is at the door to the lowest levels of the polynomial hierarchy, we cannot hope to have an encoding which nicely reflects the complexity of verification when D is a BADF in the same sense that we have shown is the case for e.g. the encoding of credulous reasoning for the admissible semantics (Section 3.1.3.3). Nevertheless, when D is a BADF the encoding can indeed be further simplified giving us the Π_1 QBF:

$$
\begin{aligned}
Ver_{grd}(D,v) &\equiv\ \forall V'''(\Psi \wedge X) \\
\Psi &:= (\psi \wedge \psi')\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)]\}\}^V) \\
X &:= \chi[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P^3_S/v(S)] \\
V &:= V_1 \ldots V_n \\
V''' &:= P'^3_S\ V'_1 \ldots V'_n \\
V_i &:= R^{s'_i}{}_{<s_i,par_D(s_i))>}\ R^{s''_i}{}_{<s_i,par_D(s_i))>}\ R^{s'''_i}{}_{<s_i,par_D(s_i))>}R^{s''''_i}{}_{<s_i,par_D(s_i))>} \quad (1 \leq i \leq n) \\
V'_i &:= T^{s'_i}{}_{<s_i,par_D(s_i))>}\ T^{s''_i}{}_{<s_i,par_D(s_i))>}\ T^{s'''_i}{}_{<s_i,par_D(s_i))>}\ T^{s''''_i}{}_{<s_i,par_D(s_i))>} \quad (1 \leq i \leq n)
\end{aligned}
$$

where ψ, ψ', and χ are defined as in the original form of the encoding for $Ver_{grd}(D,v)$.

Turning to the encodings for credulous and skeptical acceptance w.r.t. the grounded semantics, rather than making direct use of the defining encoding function $\mathcal{E}'''_{grd}$, we make use of a simple consequence that follows from the characterisation of the grounded semantics as the minimally-informative of the candidates for the grounded interpretation as characterised by properties i)'-iii)' detailed above. The fact in question is that when $v \in grd(D)$ for some ADF D and $v(s) = x$ for some statement s of D and $x \in \{\mathbf{t}, \mathbf{f}\}$, then $v'(s) = x$ has to be the case for any v' that is a candidate for being the grounded interpretation. The reason is simply that if $v'(s) \neq x$ for some candidate for the grounded interpretation v', then it is not the case that $v(s) \leq_i v'(s)$ and, hence, v is not the minimally informative candidate.

The fact we alluded means that whenever $Cred_{grd}(D,s^*) = Skept_{grd}(D,s^*) =$ *yes*, it must be the case that $v'(s) = \mathbf{t}$ for any v' that is a candidate for being the grounded interpretation of D. Our encoding for credulous and skeptical acceptance expresses this in QBF terms:

$$\begin{aligned}
&Cred_{grd}(D, s^*) \equiv Skept_{grd}(D, s^*) \equiv \ \psi \wedge \chi \\
&\quad \psi := links[D, A_{L_D}, O_{L_D}, P_S, P'_S] \\
&\quad \chi := \forall P^3_S((coh[P^3_S] \wedge prop2[D, P^3_S, A_{L_D}, O_{L_D}, R_{<s,par_D(s))>}, R'_{<s,par_D(s))>}]) \rightarrow p_{s^*}{}^{\oplus}).
\end{aligned}$$

In prenex normal form we obtain:

$$\begin{aligned}
&Cred_{grd}(D, s^*) \equiv Skept_{grd}(D, s^*) \equiv \ \exists V \ \forall V' \ (\psi \wedge \chi) \\
V &:= A_{L_D} O_{L_D} P_{<L^?_D,\exists,1>} P'_{<L^?_D,\exists,2>} \\
V' &:= P_{<L^?_D,\forall,1>} P'_{<L^?_D,\forall,2>} P^3_S \ V_1 \ldots V_n \\
V_i &:= R^{s'_i}{}_{<s_i,par_D(s_i))>} \ R^{s''_i}{}_{<s_i,par_D(s_i))>} \ R^{s'''_i}{}_{<s_i,par_D(s_i))>} \ R^{s''''_i}{}_{<s_i,par_D(s_i))>} \quad (1 \leq i \leq n) \\
\psi &:= links^M[D, A_{L_D}, O_{L_D}, P_{<L^?_D,\exists,1>}, P'_{<L^?_D,\exists,2>}, P_{<L^?_D,\forall,1>}, P'_{<L^?_D,\forall,2>}] \\
\chi &:= ((coh[P^3_S] \wedge prop2^M[D, P^3_S, A_{L_D}, O_{L_D}, V_1{}^{\mathcal{L}}, \ldots, V_n{}^{\mathcal{L}}]) \rightarrow p_{s^*}{}^{\oplus}).
\end{aligned}$$

When $L^!_D = L_D$, the encoding simplifies to the Π_1 QBF

$$\begin{aligned}
&Cred_{grd}(D, s^*) \equiv Skept_{grd}(D, s^*) \equiv \ \forall V' \ (\psi \wedge \chi)[A_{L^!_D} \cup O_{L^!_D} / \iota_D(L_D)] \\
V' &:= P^3_S \ V_1 \ldots V_n \\
V_i &:= R^{s'_i}{}_{<s_i,par_D(s_i))>} \ R^{s''_i}{}_{<s_i,par_D(s_i))>} \ R^{s'''_i}{}_{<s_i,par_D(s_i))>} \ R^{s''''_i}{}_{<s_i,par_D(s_i))>} \quad (1 \leq i \leq n)
\end{aligned}$$

thus correctly capturing the complexity of credulous and skeptical acceptance for the grounded semantics which is coNP-complete. Just as for the verification problem, we already have a “too easy” encoding to be able to further capture at the QBF level the complexity of ADF reasoning e.g. when D is a BADF; nevertheless, it should be easy for the reader to see that if one has guessed an interpretation $v^?$ for the statements S of D, then the whole encoding can be simplified (via propagation on the variables in the sets $V_1 \ldots V_n$) to a formula where all variables have been replaced for the truth constants $\bot$ and $\top$.

3.1.3.7 Encodings for the stable semantics

We conclude our presentation of link information sensitive encodings for ADFs by giving such an encoding also for the stable semantics. As hinted at in Section 3.1.2, our link information sensitive encoding follows the same pattern as the non link information sensitive encoding $\mathcal{E}''_{stb}$ we presented in that section.

The difference between the link information sensitive defining encoding function for the stable semantics and the defining encoding function $\mathcal{E}''_{stb}$ is, first of all, that the link information sensitive encoding makes use of the module $links[.,.,.,.,.]$ (Section 3.1.3.2). Secondly, the link information sensitive version makes use of the module $prop2[.,.,.,.,.,.]$ rather than the module $prop[.,.,.,.]$ to encode the properties i)-iii) (equivalently, i')-iii'); see Section 3.1.3.6) that candidates for the grounded interpretation need to satisfy. We thus arrive at the following defining encoding function:

$$\begin{aligned}
&\mathcal{E}'''_{stb}[D, P_S, A_{L_D}, O_{L_D}, P'_S, P''_S, R_{<S,par_D(S))>}, R'_{<S,par_D(S))>}] := \psi \wedge \psi' \\
&\qquad \psi := \mathcal{E}_{mod}[D, P_S] \wedge links[D, A_{L_D}, O_{L_D}, P'_S, P''_S] \\
&\qquad \psi' := \forall P'^3_S((coh[P'^3_S] \wedge \psi'') \rightarrow_{\leq_i} [P_S, P'^3_S]) \\
&\qquad \psi'' := prop2[(S, \{\phi_s \wedge p_s\}_{s \in S}), P'^3_S, A_{L_D}, O_{L_D}, R_{<S,par_D(S))>}, R'_{<S,par_D(S))>}]).
\end{aligned}$$

Proposition 9 states the fact that $\mathcal{E}'''_{stb}$-, just as its non link information sensitive counterpart $\mathcal{E}''_{stb}$ (derived from the defining encoding function $\mathcal{E}_{stb}$, see Section 3.1.2) ,- correctly captures the stable semantics in QBF terms.

Proposition 9. *Given an ADF $D = (S, \{\phi_s\}_{s \in S})$ and the indexed variables P_S, A_{L_D}, O_{L_D}, P'_S, P''_S, $R_{<S,par_D(S))>}$, $R'_{<S,par_D(S))>}$, let $\mathcal{E}'''_{stb}$ be the function returning the QBF with free variables in P_S, $\mathcal{E}''_{stb}[D, P_S, A_{L_D}, O_{L_D}, P'_S, P''_S, R_{<S,par_D(S))>}, R'_{<S,par_D(S))>}]$. Then $\mathcal{E}'''_{stb}$ is a defining encoding function for the stable semantics.*

Proof. (sketch) The proof is similar to that of Proposition 3 in Section 3.1.2, the main difference being that now also Proposition 7 needs to be used and the proof of Proposition 3 needs to be adapted to the use of the module $prop2[.,.,.,.,.,.]$ rather than $prop[.,.,.,.]$. □

Turning to the encodings for the reasoning tasks, the link information sensitive encoding for the verification problem based on the defining encoding function $\mathcal{E}'''_{stb}$ is

$$\begin{aligned}
&\qquad Ver_{stb}(D, v) \equiv\ \exists V \forall V'(\psi \wedge \psi')[P_S/v(P_S)] \\
&V := A_{L_D} O_{L_D} P_{<L^?_D,\exists,1>} P'_{<L^?_D,\exists,2>} \\
&V' := P_{<L^?_D,\forall,1>} P'_{<L^?_D,\forall,2>} P'^3_S\ V'_1 \dots V'_n \\
&V'_i := R^{s'_i}{}_{<s_i,par_D(s_i))>}\ R^{s''_i}{}_{<s_i,par_D(s_i))>}\ R^{s'''_i}{}_{<s_i,par_D(s_i))>}\ R^{s''''_i}{}_{<s_i,par_D(s_i))>} \quad (1 \leq i \leq n) \\
&\psi := \mathcal{E}_{mod}[D, P_S] \wedge links^M[D, A_{L_D}, O_{L_D}, P_{<L^?_D,\exists,1>}, P'_{<L^?_D,\exists,2>}, P_{<L^?_D,\forall,1>}, P'_{<L^?_D,\forall,2>}] \\
&\psi' := ((coh[P'^3_S] \wedge \psi'') \rightarrow_{\leq_i} [P_S, P'^3_S])
\end{aligned}$$

$$\psi'' := prop2^M[(S, \{\phi_s \wedge p_s\}_{s\in S}), P'^3_S, A_{L_D}, O_{L_D}, V'^{\mathcal{L}}_1, \dots, V'^{\mathcal{L}}_n].$$

When all of the types of the links of the ADF D are known, the encoding can be simplified

$$\begin{aligned}
Ver_{stb}(D,v) &\equiv\ \forall V'(\psi \wedge \psi')[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P_S/v(S)] \\
V' &:= P'^3_S\ V'_1 \dots V'_n \\
V'_i &:= R^{s'_i}{}_{<s_i, par_D(s_i))>}\ R^{s''_i}{}_{<s_i, par_D(s_i))>}\ R^{s'''_i}{}_{<s_i, par_D(s_i))>}\ R^{s''''_i}{}_{<s_i, par_D(s_i))>} \quad (1 \leq i \leq n)
\end{aligned}$$

thus giving us a Π_1 encoding that matches the complexity of the verification problem for the stable semantics (coNP-complete). Furthermore, when D is a BADF, for the negated dual version of the encoding,- propagation on the variables $V'_1 \dots V'_n$ when given a candidate for a counter-example $v^?$ showing that $v \notin grd(D^v_*)$ gives us

$$\begin{aligned}
Ver_{stb}(D,v) &\equiv\ \Psi\{\{[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P'_S/v^?]\}\}^{V'} \\
\Psi &:= \neg(\psi \vee \psi')[A_{L^!_D} \cup O_{L^!_D}/\iota_D(L_D)][P_S/v(S)] \\
V' &:= V'_1 \dots V'_n \\
V'_i &:= R^{s'_i}{}_{<s_i, par_D(s_i))>}\ R^{s''_i}{}_{<s_i, par_D(s_i))>}\ R^{s'''_i}{}_{<s_i, par_D(s_i))>}\ R^{s''''_i}{}_{<s_i, par_D(s_i))>} \quad (1 \leq i \leq n)
\end{aligned}$$

(here ψ and ψ' are defined as above), i.e. an encoding where all variable have been replaced with truth constants $\bot$ and $\top$. This reflects at the QBF level the complexity for the verification problem for the stable semantics when D is a BADF. Specifically, in that the complexity stems from guessing that a counterexample for showing $v \notin grd(D^v_*)$, but not from checking that the counterexample is a candidate for being a grounded interpretation.

Turning to credulous acceptance, an encoding based on the defining encoding function $\mathcal{E}'''_{stb}$ is

$$\begin{aligned}
Cred_{stb}(D,s^*) &\equiv\ \exists V \forall V'(\psi \wedge \psi' \wedge p_{s^*}) \\
V &:= P_S A_{L_D} O_{L_D} P_{<L^?_D,\exists,1>} P'_{<L^?_D,\exists,2>} \\
V' &:= P_{<L^?_D,\forall,1>} P'_{<L^?_D,\forall,2>} P'^3_S\ V'_1 \dots V'_n \\
V'_i &:= R^{s'_i}{}_{<s_i, par_D(s_i))>}\ R^{s''_i}{}_{<s_i, par_D(s_i))>}\ R^{s'''_i}{}_{<s_i, par_D(s_i))>}\ R^{s''''_i}{}_{<s_i, par_D(s_i))>} \quad (1 \leq i \leq n) \\
\psi &:= \mathcal{E}_{mod}[D, P_S] \wedge links^M[D, A_{L_D}, O_{L_D}, P_{<L^?_D,\exists,1>}, P'_{<L^?_D,\exists,2>}, P_{<L^?_D,\forall,1>}, P'_{<L^?_D,\forall,2>}] \\
\psi' &:= ((coh[P'^3_S] \wedge \psi'') \rightarrow_{\leq_i} [P_S, P'^3_S]) \\
\psi'' &:= prop2^M[(S, \{\phi_s \wedge p_s\}_{s\in S}), P'^3_S, A_{L_D}, O_{L_D}, V'^{\mathcal{L}}_1, \dots, V'^{\mathcal{L}}_n].
\end{aligned}$$

This encoding returns a Σ_2 encoding, thus already matching the complexity of credulous acceptance for ADFs (Σ_2^P-complete). When D is a BADF and we have a guess for the stable semantics $v^?$, the encoding can again be simplified giving us

$$
\begin{aligned}
&\quad\quad Cred_{stb}(D, s^*) \equiv\ \forall V'(\psi \wedge \psi' \wedge p_{s^*})[A_{L_D^!} \cup O_{L_D^!}/\iota_D(L_D)][P_S/v^?(S)]\\
V &:= P_S\\
V' &:= {P'}_S^3\ V'_1 \dots V'_n\\
V'_i &:= R^{s'_i}{}_{<s_i, par_D(s_i))>}\ R^{s''_i}{}_{<s_i, par_D(s_i))>}\ R^{s'''_i}{}_{<s_i, par_D(s_i))>}\ R^{s''''_i}{}_{<s_i, par_D(s_i))>} \quad (1 \leq i \leq n)
\end{aligned}
$$

with the dual form, i.e.

$$\exists V'(\neg\psi \vee \neg\psi' \vee \neg p_{s^*})[A_{L_D^!} \cup O_{L_D^!}/\iota_D(L_D)][P_S/v^?(S)]$$

, being a Σ_1-QBF, thus reflecting the complexity of credulous acceptance for a BADF (NP-complete).

For skeptical acceptance w.r.t. the stable semantics, we can construct an encoding from $\mathcal{E}'''_{stb}$ as follows:

$$
\begin{aligned}
&\quad\quad Skept_{stb}(D, s^*) \equiv\ \forall V \exists V'((\psi \wedge \psi') \rightarrow p_{s^*})\\
V &:= P_S A_{L_D} O_{L_D} P_{<L_D^?, \exists, 1>} P'_{<L_D^?, \exists, 2>}\\
V' &:= P_{<L_D^?, \forall, 1>} P'_{<L_D^?, \forall, 2>} {P'}_S^3\ V'_1 \dots V'_n\\
V'_i &:= R^{s'_i}{}_{<s_i, par_D(s_i))>}\ R^{s''_i}{}_{<s_i, par_D(s_i))>}\ R^{s'''_i}{}_{<s_i, par_D(s_i))>}\ R^{s''''_i}{}_{<s_i, par_D(s_i))>} \quad (1 \leq i \leq n)\\
\psi &:= \mathcal{E}_{mod}[D, P_S] \wedge links^M[D, A_{L_D}, O_{L_D}, P_{<L_D^?, \exists, 1>}, P'_{<L_D^?, \exists, 2>}, P_{<L_D^?, \forall, 1>}, P'_{<L_D^?, \forall, 2>}]\\
\psi' &:= ((coh[{P'}_S^3] \wedge \psi'') \rightarrow_{\leq_i} [P_S, {P'}_S^3])\\
\psi'' &:= prop2^M[(S, \{\phi_s \wedge p_s\}_{s \in S}), {P'}_S^3, A_{L_D}, O_{L_D}, {V'_1}^{\mathcal{L}}, \dots, {V'_n}^{\mathcal{L}}].
\end{aligned}
$$

This encoding returns a Π_2 QBF, thus matching the complexity of skeptical reasoning for the stable semantics (Π_2^P-complete). A similar argument as that given for the encoding for credulous acceptance, shows that also this final encoding we give in this section reflects at the QBF level the complexity of the reasoning problem it encodes when information about the link types is available.

3.1.4 Implementation and experiments

We have initiated a re-implementation of our system QADF [Dil14, DWW14] in order to make it more modular and easier to inspect. Thus, while the system QADF in the version reported on in [DWW14] (version 0.1) directly produces the encodings in the QDIMACS[6] format that most QBF solvers expect, the new version also produces an intermediate representation in which the encoding is in prenex-normal-form but the matrix is not in CNF. This allows for easier inspection of the encodings[7] and also making use of QBF solvers that do not require the matrix to be in CNF (in fact our implementation also produces some of the encodings in the QCIR format [JKS16], but this is a feature which remains to be more extensively tested).

Our current re-implementation of QADF (version 0.4.0) is publicly available[8] and produces encodings for the existence (and enumeration) problem as well as credulous and skeptical acceptance for the admissible, preferred, and stable semantics. It also produces the link-information-sensitive encodings for these problems. It is implemented in Scala[9] and can, therefore, be run as a Java[10] executable.

The input format for QADF is the input format that has become the standard for ADF systems. Each statement x of the input ADF is encoded via the string $s(x)$ (alternatively, for legacy reasons, also $statement(x)$ can be used). The acceptance condition F of x is specified in prefix notation via $ac(x, F)$ as illustrated in Example 7. As is also shown in the example, information about the links can be added to the specification of the input using *att* and *sup* predicates.

Example 7. *The ADF from Example 1, as input to QADF, is encoded as follows:*

```
s(a).
s(b).
s(c).
ac(a,or(neg(b),b)).
ac(b,b).
ac(c,imp(c,b)).
```

Note the period at the end of each line. Here or, imp, neg stand for $\vee$, $\rightarrow$, $\neg$ respectively. On the other hand and, $c(v)$, and $c(f)$ can be used for $\wedge$, $\top$, and $\bot$. The following shows how to specify the attack and support relations for the ADF from Example 1. These are added to the part of the input specifying the ADF.

[6]http://www.qbflib.org/qdimacs.html

[7]Potentially, at the cost of some loss in efficiency in producing the encodings. For this reason and the fact that our previous system also suppports the model and complete (but not the stable) semantics we have kept the older version on the website dedicate to QADF.

[8]https://www.dbai.tuwien.ac.at/proj/adf/qadf/

[9]https://www.scala-lang.org/

[10]https://www.java.com

```
att(b,a).
sup(b,a).
sup(b,b).
sup(b,c).
att(c,c).
```

If for some link the relation is not specified, `QADF` *treats the link type as unknown. Also note that for redundant links that they are both attacking and supporting must be specified. Dependent links can be specified using the binary predicate dep. For instance,*

```
dep(b,a).
```

indicates that the link (b,a) is a dependent link.

A typical call of `QADF` (using a `UNIX` command line) looks as follows:

```
java -jar qadf_0.4.0.jar -adm -cred a -L -D filename | \
      ./path/to/bloqqer | ./path/to/depqbf
```

Here we ask for the encoding of credulous acceptance (of the statement “a”) w.r.t. the admissible semantics for the ADF specified in the file “filename” and pipe the encoding (in QDIMACS format) to the preprocessing tool `Bloqqer`[11] and the QBF-solver `DepQBF`[12]. Moreover, we ask for the link-information-sensitive version of the encoding and the dual version. The dual version is the version which returns the dual answer to the original encoding; for instance, it returns “UNSAT” when the original encoding is sastisfiable (“SAT”). The reason this is useful is that the Tseitin transformation (see Section 2.1.4) for converting the matrix of the encoding to CNF introduces a further block of (innermost existentially) quantified variables which may make the prefix type of the resulting encoding one level higher in the polynomial hierarchy in terms of its computational complexity. Asking for the dual encoding allows to circumvent this issue.

We provide the complete usage (subject to change in future versions) of `QADF`:

```
usage: qadf [options] inputfile
with options:
 -h              display this help (also works with --h, \
                                                 -help, --help)
 -version        print version
 -adm            admissible
 -prf            preferred
```

[11] `http://fmv.jku.at/bloqqer/`
[12] `http://lonsing.github.io/depqbf/`

```
  -stb            stable
  -stb2           stable (using Dung's 2018 characterisation)
  -cred s         check credulous acceptance of statement s
  -scep s         check skeptical acceptance of statement s
  -O outputfile   print output to outputfile
  -D              use dual encoding
  -L              use link information sensitive encoding
  -noTransform    do not apply any transformation to encoding
  -Tseitin        only apply tseitin transformation to encoding
  -Circuit        Output circuit representation
  -QCIR           Output circuit representation in QCIR 14 format
Default mode is print encoding of existence problem of \
selected semantics to standard output (in qdimacs format)
```

Note that there are several options for the output format and also the option for an encoding for the stable semantics based on [DT18] which are tangential to this work.

For the 2018 "international workshop on quantified boolean formulas" (QBF'18) we carried out an initial study on the effect of some of the top-ranked QBF solvers from the QBF evaluation 2017 (QBFEval'17)[13] as well as main preprocessing tools for QSAT solving on our link-information-sensitive encodings. We also considered the version of the solver `DepQBF` (developed at TU Wien) using heavy preprocessing first presented at QBFEval'18 (therefore dubbed `DepQBF'18`). We considered credulous reasoning for the admissible semantics and sceptical reasoning for the preferred semantics. Specifically, in the study we included ADFs with different percentages (20% and 60%) of dependent links as well as different percentages (0%, 25%, 50%, and 75%) of the link types to be unknown. The ADFs are generated on the basis of graphs representing transportation networks. We compared with the performance of the QSAT solvers and preprocessors we considered on the non-link-information-sensitive encodings from [Dil14, DWW14].

For details and the results of our study see Appendix A. Confirming results in other studies [DWW14, BDH$^+$17, Kes17, DKLW18, LMN$^+$18a] in which the non-link-information-sensitive encodings produced by our system `QADF` have been evaluated, the results for the preferred semantics are rather disappointing. Thus, the best performing solver and preprocessor combination (`QADF` with `dynDepQBF` + `Bloqqer`) solves around 24% of the non-link-information-sensitive encodings and 21% of the link-information-sensitive encodings (time-out of 1800 seconds). In general, the solvers perform somewhat worse on the link-information-sensitive encodings; although the use of pre-processing levels-out the performance in some cases (e.g. combination of either `dynDepQBF` or `DepQBF` with the preprocessor `HQSpre`).

Also confirming other studies, the results for the admissible semantics are more promising. Thus the best performing solver for this semantics (`DepQBF`'18; i.e. the version of

[13]`http://www.qbflib.org/event_page.php?year=2017`

DepQBF submitted to QBFEval'18) solves around 99% of the non-link-information-sensitive encodings and around 97% of the link-information-sensitive-encodings (with average solving times under a minute and two minutes respectively). Interestingly, the preprocessors we considered in the study (Bloqqer and HQSpre) seem to latch on to the information provided by the links. Thus, for instance, Bloqqer used as a stand-alone-tool is able to solve e.g. around 77% of the link-information-sensitive encodings (in under 3 seconds) for ADFs with 20% dependent links vs. 67% of the non-link-information-sensitive encodings. Although for the best performing tool (DepQBF'18) this is not the case, for several of the solvers the use of preprocessing then improves their performance on the link-information-sensitive vs. non-link-information sensitive encodings.

To conclude, at least on the ADFs we considered in our study for QBF'18, reasoning for the preferred semantics does not seem to be eased by information about the structure of ADFs. On the other hand, for the admissible semantics, such information does improve the performance of some systems (when used with preprocessors). This is consistent with studies on the performance of QADF (among other systems and considering only the non-link-information-sensitive encodings) on acyclic (ADFs having an underlying acyclic graph) vs. non acyclic ADFs carried out in [DKLW18], where there are gains in performance on the acyclic instances for the admissible semantics but no such gains in performance can be observed for the preferred semantics. On the other hand, the impressive gains (also comparing to the best performing versions of DepQBF in previous studies) obtained by the use of the best performing QSAT solver (DepQBF'18) on the admissible semantics in our study for QBF'18, do not seem to arise from the information about the links. We refer to Section B.3 in Appendix B for a survey of all recent empirical evaluations of ADF systems (including the non-link-information-sensitive encodings produced by QADF) and further discussion of the results of our evaluation for QBF'18 in light thereof.

3.2 Dynamic ASP encodings for ADFs

In this section we demonstrate, for purposes of providing ASP encodings for ADF reasoning, the use of a fact about the complexity of ASP programs. The fact in question is that the combined complexity of ASP for programs with predicates of bounded arity [EFFW07], just as the complexity of many of the acceptance problems for ADFs, occupies the second and third level of the polynomial hierarchy (see sections 2.3.3 and 2.2.4). This allows for dynamic yet single shot encodings to fragments of ASP with matching complexity.

Similarly to our QBF encodings, we here construct ASP encodings π_σ for the semantics $\sigma \in \{adm, com, prf, grd, stb\}$ such that there is a certain one to one correspondence between the σ interpretations of an ADF $D = (S, \{\phi_s\}_{s \in S})$ and the answer sets of $\pi_\sigma(D)$ (the encoding function π_σ applied to D). More precisely, we will use atoms $asg(s, x)$ with $s \in S, x \in \{1, 0, \mathbf{u}\}$ to represent ADF interpretations in our encodings. Throughout this section we represent the truth values true and false with 0 and 1 rather than **f** and **t** for technical reasons: in the encodings we make use of ASP built in binary arithmetic functions and comparison predicates for evaluating propositional formulas such as ADF acceptance conditions.

An interpretation v of D and a set of ground atoms (interpretation of an ASP program) I correspond to each other, $v \cong I$, whenever for every $s \in S$, $v(s) = x$ iff $asg(s, x) \in I$. We overload $\cong$ to get the correspondence between sets of interpretations and sets of answer sets we aim for.

Definition 17. *Given a set of (ADF) interpretations V and a collection of sets of ground atoms (ASP interpretations) $\mathcal{I}$, we say that V and I correspond, $V \cong \mathcal{I}$, if*

1. *for every $v \in V$ there is an $I \in \mathcal{I}$ s.t. $v \cong I$;*
2. *for every $I \in \mathcal{I}$ there is a $v \in V$ s.t. $v \cong I$.*

Having encodings π_σ for $\sigma \in \{adm, com, prf, grd, stb\}$ for which $\sigma(D) \cong \mathcal{AS}(\pi_\sigma(D))$ for any ADF D means we have encodings for the enumeration problem for the different semantics. Encodings for credulous and skeptical reasoning are obtained via the homonymous ASP reasoning tasks applied on the encodings for the enumeration problem.

3.2.1 Encodings for the admissible semantics

In the course of presenting our dynamic ASP encodings for the admissible semantics we introduce several elements we will make use of throughout Section 3.2. Among these is that all encodings will assume a simple set of facts indicating the statements of the input ADF $D = (S, \{\phi_s\}_{s \in S})$:

$$\pi_{arg}(D) := \{arg(s).\mid s \in S\}.$$

Also, several of the encodings will need facts for encoding the possible truth values that can be assigned to a statement s by a completion of an interpretation mapping s to $\mathbf{u}$, 1, and 0, respectively:

$$\pi_{lt} := \{lt(\mathbf{u}, 0).\ lt(\mathbf{u}, 1).\ lt(1, 1).\ lt(0, 0).\}.$$

All of our encodings, including the one for the admissible semantics, follow the guess & check methodology that, as stated in Section 2.2, is at the heart of the ASP paradigm [JN16]. Here parts of a program delineate candidates for a solution to a problem. These are often referred to as “guesses”. Other parts of the program, the “constraints”, then check whether the guessed candidates are indeed solutions. In the case of the encodings for ADFs the guessing part of the programs outline possible assignments of truth values to the statements, i.e. an ADF interpretation. For the three valued semantics, as the admissible semantics, the rules are as follows:

$$\begin{aligned}\pi_{guess} := \{ & asg(S,0) \,\texttt{:-}\, not\ asg(S,1), not\ asg(S,\mathbf{u}), arg(S).\\ & asg(S,1) \,\texttt{:-}\, not\ asg(S,\mathbf{u}), not\ asg(S,0), arg(S).\\ & asg(S,\mathbf{u}) \,\texttt{:-}\, not\ asg(S,0), not\ asg(S,1), arg(S).\}.\end{aligned}$$

In all our encodings we will need to encode the semantic evaluation of propositional formulas; e.g. the evaluation of the acceptance conditions by completions of an interpretation. Given a propositional formula ϕ, for this we introduce the function Ω. For assignments of truth values (1 and 0) to the propositional variables in ϕ, $\Omega(\phi)$ gives us a set of atoms corresponding to the propagation of the truth values to the subformulas of ϕ in accordance with the semantics of classical propositional logic. The atoms make use of ASP variables V_ψ where ψ is a subformula of ϕ. The variables V_p, where p is a propositional variable occurring in ϕ, can be used by other parts of ASP rules employing the atoms in $\Omega(\phi)$ for purposes of assigning intended truth values to the propositional variables in ϕ.

For the definition of the atoms $\Omega(\phi)$ we rely on the ASP built in arithmetic functions `&` (bitwise AND), `?` (bitwise OR), and `-` (subtraction). We also use the built in comparison predicate = (see Section 2.2.3). Let ϕ be a propositional formula over a set of propositional variables P; then the relevant set of atoms is defined as

$$\Omega(\phi) := \begin{cases} \Omega(\phi_1) \cup \Omega(\phi_2) \cup \{V_\phi = V_{\phi_1} \,\texttt{\&}\, V_{\phi_2}\} & \text{if } \phi = \phi_1 \wedge \phi_2 \\ \Omega(\phi_1) \cup \Omega(\phi_2) \cup \{V_\phi = V_{\phi_1} \,\texttt{?}\, V_{\phi_2}\} & \text{if } \phi = \phi_1 \vee \phi_2 \\ \Omega(\psi) \cup \{V_\phi = 1\texttt{-}V_\psi\} & \text{if } \phi = \neg\psi \\ \emptyset & \text{if } \phi = p \in P \end{cases}$$

where V_ϕ, V_{ϕ_1} V_{ϕ_2} and V_ψ are variables representing the subformulas of ϕ.

Our encoding for the admissible semantics, π_{adm}, is based on the fact that an interpretation v for an ADF $D = (S, \{\phi_s\}_{s\in S})$ is admissible iff for every $s \in S$ it is the case that

- if $v(s) = 1$ then there is no $w \in [v]_2$ s.t. $w(\phi_s) = 0$,
- if $v(s) = 0$ then there is no $w \in [v]_2$ s.t. $w(\phi_s) = 1$.

This is a simple consequence of the definition of the admissible semantics (Definition 8). Any $w \in [v]_2$ which contradicts this simple observation (e.g. $v(s) = 1$ and $w(\phi_s) = 0$) is a “counter-model” to v being an admissible interpretation. The constraining part of our encoding for the admissible semantics essentially disallows guessed assignments of truth values to the statements of an ADF corresponding to ADF interpretations which have counter-models to them being admissible.

To encode the constraints of our encoding we need auxiliary rules firing when the guessed assignments have counter-models to them being admissible. These rules, two for each

$s \in S$, make use of bodies ω_s where $\Omega(\phi_s)$ is employed to evaluate the acceptance conditions by the completions. The latter are obtained by setting variables V_t for $t \in par_D(s)$ with the adequate truth values by using the predicates *asg* and *lt* defined in π_{guess} and π_{lt} respectively:

$$\omega_s := \{asg(t, Y_t), lt(Y_t, V_t) \mid t \in par_D(s)\} \cup \Omega(\phi_s).$$

The two rules for every statement $s \in S$ have heads $sat(s)$ and $inv(s)$ that fire in case there is some completion of the interpretation corresponding to the assignments guessed in the program fragment π_{guess} such that the acceptance condition ϕ_s evaluates to 1 and 0, respectively:

$$\begin{aligned}\pi_{sat}(D) := \{&sat(s)\,\text{:-}\,\omega_s, V_{\phi_s} = 1.\\ &inv(s)\,\text{:-}\,\omega_s, V_{\phi_s} = 0. \mid s \in S\}.\end{aligned}$$

The encoding for the admissible semantics now results from compounding the program fragments $\pi_{arg}(D)$, π_{lt}, π_{guess}, and $\pi_{sat}(D)$ together with ASP constraints which filter out assignments corresponding to interpretations of D having counter-models to being admissible.

$$\begin{aligned}&\pi_{adm}(D) := \pi_{arg}(D) \;\cup\; \pi_{lt} \;\cup\; \pi_{guess} \;\cup\; \pi_{sat}(D) \;\cup\\ &\{\,\text{:-}\,arg(S), asg(S,1), inv(S).\;\; \text{:-}\,arg(S), asg(S,0), sat(S).\}.\end{aligned}$$

Proposition 10 formally states that π_{adm} is indeed an adequate encoding function. For the proof, which is prototypical for most of the proofs of correctness in Section 3.2, we use the notation

$$I_p := \{p(t_1, \dots, t_n) \in I\}.$$

For an ASP interpretation I (set of ground atoms), I_p represents I projected onto the predicate p (with arity n).

Proposition 10. *For every ADF D it holds that $adm(D) \cong \mathcal{AS}(\pi_{adm}(D))$.*

Proof. Let $D = (S, \{\phi_s\}_{s \in S})$ be an ADF and $v \in adm(D)$. Let also

$$I := \{arg(s) \mid s \in S\} \cup$$

$$\begin{aligned}&\{\mathit{lt}(\mathbf{u},0), \mathit{lt}(\mathbf{u},1), \mathit{lt}(1,1), \mathit{lt}(0,0)\} \cup\\ &\{\mathit{asg}(s,x) \mid s \in S, v(s)=x\} \cup\\ &\{\mathit{sat}(s) \mid \textit{if there is a } w \in [v]_2 \textit{ s.t. } w(\phi_s)=1\} \cup\\ &\{\mathit{inv}(s) \mid \textit{if there is a } w \in [v]_2 \textit{ s.t. } w(\phi_s)=0\}\end{aligned}$$

be a set of ground atoms (such that $v \cong I$). We prove now that $I \in \mathcal{AS}(\pi_{adm}(D))$.

We start by proving that I satisfies $\pi_{adm}(D)^I$. First note that I satisfies $\pi_{arg}(D)^I = \pi_{arg}(D)$ as well as $\pi_{lt}^I = \pi_{lt}$ since all the atoms making up the facts in these two modules are in I (first two lines of the definition of I). I also satisfies

$$\begin{aligned}\pi_{guess}^I =&\{\mathit{asg}(s,x) \text{ :- } \mathit{arg}(s). \mid s \in S,\\ &\mathit{asg}(s,y) \notin I, \mathit{asg}(s,z) \notin I, x \in \{1,0,\mathbf{u}\}, y,z \in (\{1,0,\mathbf{u}\} \setminus \{x\})\}\end{aligned}$$

since, first of all, $\mathit{arg}(s) \in I$ iff $s \in S$ by the first line of the definition of I (and the fact that the predicate *arg* does not appear in the head of any rules other than $\pi_{arg}(D)^I = \pi_{arg}(D)$). Secondly, for any $s \in S$, $\mathit{asg}(s,x) \in I$ whenever $\mathit{asg}(s,y) \notin I$ and $\mathit{asg}(s,z) \notin I$ for $x \in \{1,0,\mathbf{u}\}$ and $y,z \in (\{1,0,\mathbf{u}\} \setminus \{x\})$ by the fact that $v \cong I$ (third line of the definition of I).

Now consider the rule $r \in \pi_{sat}(D)$ with $H(r) = \mathit{sat}(S)$ and a substitution σ s.t. $\sigma r \in \pi_{sat}(D)^I$. This means that σr is of the form

$$\mathit{sat}(s)\text{:-}\sigma\omega_s, \sigma(V_{\phi_s}=1).$$

with

$$\sigma\omega_s = \{\mathit{asg}(t,y_t), \mathit{lt}(y_t,v_t) \mid t \in \mathit{par}_D(s)\} \cup \sigma\Omega(\phi_s)$$

and where $\sigma(Y_t) = y_t$, $\sigma(V_t) = v_t$. If $B(\sigma r) \in I$, it must be the case that $y_t \in \{1,0,\mathbf{u}\}$, $v_t \in \{0,1\}$ for $t \in \mathit{par}_D(s)$ and $\sigma\Omega(\phi_s) \in I$. Now it should be easy for the reader to see that from the fact that $\{\mathit{asg}(t,y_t), \mathit{lt}(y_t,v_t) \mid t \in \mathit{par}_D(s)\} \subseteq I$ and $v \cong I$ it is the case that $w \in [v]_2$ for the ADF interpretation w defined as $w(t) = v_t$ for every $t \in \mathit{par}_D(s)$. It is also simple to establish that $\sigma\Omega(\phi_s) \in I$ and $\sigma(V_{\phi_s}=1) \in I$ imply that $w(\phi_s) = 1$. Hence, by the fourth line of the definition of I $\mathit{sat}(s) \in I$, i.e. I satisfies σr. In the same manner, by the fifth line of the definition of I it follows that I satisfies any grounding $\sigma r \in \pi_{sat}(D)^I$ for the rule r s.t. $H(r) = \mathit{inv}(S)$. In conclusion, I satisfies $\pi_{sat}(D)^I$.

Let us turn now to a ground instance $r \in \pi_{adm}(D)^I$

$$:\text{-}\ arg(s), asg(s,0), sat(s).$$

of the constraint

$$:\text{-}\ arg(S), asg(S,0), sat(S).$$

$\in \pi_{adm}(D)$. By the fourth line of the definition of I $sat(s) \in I$ iff there is a $w \in [v]_2$ s.t. $w(\phi_s) = 1$. But then by the fact that $v \in adm(D)$ and $v \cong I$, $asg(s,0) \notin I$, i.e. r can not be satisfied by I. In the same manner also any ground instance in $\pi_{adm}(D)^I$ of the constraint

$$:\text{-}\ arg(S), asg(S,1), inv(S).$$

can not be satisfied by I.

We have established that I satisfies $\pi_{adm}(D)^I$. We continue our proof of $I \in \mathcal{AS}(\pi_{adm}(D))$ by now showing that there is no $I' \subset I$ that satisfies $\pi_{adm}(D)^I$.

In effect, consider any other I' that satisfies $\pi_{adm}(D)^I$. Note first of all that then $I'_{arg} \supseteq I_{arg}$ and $I'_{lt} \supseteq I_{lt}$ because both I' and I satisfy $\pi_{arg}(D)^I$ as well as π^I_{lt}. Hence also $I'_{asg} \supseteq I_{asg}$ because I' satisfies π^I_{guess} (see the proof of I satisfies $\pi_{adm}(D)^I$ for the structure of π^I_{guess}) and $I'_{arg} \supseteq I_{arg}$, i.e. $B(r) \subseteq I'$ for every $r \in \pi^I_{guess}$. But then, since $I'_{arg} \supseteq I_{arg}$ and $I'_{asg} \supseteq I_{asg}$, and I' satisfies all the comparison predicates with arithmetic functions that I does by definition, I' satisfies all the rules in $\pi_{sat}(D)^I$ that I does (see again the proof of I satisfies $\pi_{adm}(D)^I$ for the form of such rules). Hence, also $I'_{sat} \supseteq I_{sat}$ and $I'_{inv} \supseteq I_{inv}$. In conclusion, $I' \supseteq I$.

Since I' was general we derive that there is no $I' \subset I$ that satisfies $\pi_{adm}(D)^I$. Together with the fact that I satisfies $\pi_{adm}(D)^I$ we have that $I \in \mathcal{AS}(\pi_{adm}(D))$.

We now turn to proving that for any $I \in \mathcal{AS}(\pi_{adm}(D))$ it holds that $v \in adm(D)$ for $v \cong I$. Note first that for such an I, since I satisfies $\pi_{arg}(D)^I = \pi_{arg}(D)$ as well as

$$\begin{aligned}\pi^I_{guess} = \{ & asg(s,x)\ :\text{-}\ arg(s). \mid s \in S,\\ & asg(s,y) \notin I, asg(s,z) \notin I, x \in \{1,0,\mathbf{u}\}, y,z \in (\{1,0,\mathbf{u}\} \setminus \{x\})\},\end{aligned}$$

for every $s \in S$ there is a $x \in \{1,0,\mathbf{u}\}$ such that $asg(s,x) \in I$. Also, $asg(s,x) \in I$ whenever $asg(s,y) \notin I$ and $asg(s,z) \notin I$ for $y,z \in (\{1,0,\mathbf{u}\} \setminus \{x\})$. I.e. v s.t. $v \cong I$ is well defined.

Now assume that $v \notin adm(D)$. Then there are $s \in S$, $w \in [v]_2$ for which either i) $v(s) = 1$ and $w(\phi_s) = 0$ or ii) $v(s) = 0$ and $w(\phi_s) = 1$. Let us consider the case i). In that case consider a substitution σ for the rule $r \in \pi_{sat}(D)$

$$inv(s) \,\text{:-}\, \Omega_s, V_{\phi_s} = 0.$$

where

$$\Omega_s = \{asg(t, Y_t), lt(Y_t, V_t) \mid t \in par_D(s)\} \cup \Omega(\phi_s).$$

The substitution σ is defined as $\sigma(Y_t) = v(t)$ and $\sigma(V_t) = w(t)$ for every $t \in par_D(s)$. Since $v \cong I$ we have that $asg(t, \sigma(Y_t)) \in I$ for every $t \in par_D(s)$. Also $lt(\sigma(Y_t), \sigma(V_t)) \in I$ since I satisfies π^I_{lt}.

Now, by definition $\sigma\Omega(\phi_s) \subseteq I$ and from $w(\phi_s) = 0$ it is easy to see that it follows that also $\sigma(V_{\phi_s} = 0) \in I$, i.e. $\sigma r \in \pi_{sat}(D)^I$ and $B(\sigma r) \subseteq I$. This means that also $inv(s) \in I$. As a consequence we have that $B(r') \subseteq I$ for the constraint r'

$$\text{:-}\ arg(s), asg(s, 1), inv(s).$$

in $\pi_{adm}(D)^I$. This is a contradiction to $I \in \mathcal{AS}(\pi_{adm}(D))$. From the case ii) $v(s) = 0$ and $w(\phi_s) = 1$ a contradiction can be derived in analogous manner. Hence, $v \in adm(D)$ must be the case. □

Example 8. *Considering the ADF D from Example 1, $\pi_{adm}(D)$ (as implemented by our system* `YADF` *with minor formatting for purposes of readability; see Section 3.2.7) looks as follows:*

```
arg(a).
arg(b).
arg(c).
leq(u,0).
leq(u,1).
leq(0,0).
leq(1,1).
asg(S,u) :- arg(S),not asg(S,0),not asg(S,1).
asg(S,0) :- arg(S),not asg(S,1),not asg(S,u).
asg(S,1) :- arg(S),not asg(S,u),not asg(S,0).
sat(a) :- asg(b,Y0),leq(Y0,V0),V1=1-V0,V2=V1?V0,V2=1.
sat(b) :- asg(b,Y0),leq(Y0,V0),V0=1.
sat(c) :- asg(c,Y0),leq(Y0,V0),asg(b,Y1),leq(Y1,V1),
                                        V3=1,V3=V2?V1,V2=1-V0.
inv(a) :- asg(b,Y0),leq(Y0,V0),V1=1-V0,V2=V1?V0,V2=0.
inv(b) :- asg(b,Y0),leq(Y0,V0),V0=0.
inv(c) :- asg(c,Y0),leq(Y0,V0),asg(b,Y1),leq(Y1,V1),
                                        V3=V2?V1,V3=0,V2=1-V0.
:- arg(S),asg(S,1),inv(S).
:- arg(S),asg(S,0),sat(S).
```

A possible output of an ASP solver (the current one is the simplified output of `clingo` *version 4.5.4) given this instance looks as follows (only showing asg, sat, and inv predicates):*

```
Answer: 1
asg(c,u) asg(b,0) asg(a,u) sat(c) sat(a) inv(c) inv(b)
Answer: 2
asg(c,u) asg(b,0) asg(a,1) sat(c) sat(a) inv(c) inv(b)
Answer: 3
asg(c,u) asg(b,1) asg(a,u) sat(b) sat(c) sat(a)
Answer: 4
asg(c,u) asg(b,1) asg(a,1) sat(b) sat(c) sat(a)
Answer: 5
asg(c,u) asg(b,u) asg(a,1) sat(b) sat(c) sat(a) inv(c) inv(b)
Answer: 6
asg(c,u) asg(b,u) asg(a,u) sat(b) sat(c) sat(a) inv(c) inv(b)
Answer: 7
asg(c,1) asg(b,1) asg(a,u) sat(b) sat(c) sat(a)
Answer: 8
asg(c,1) asg(b,1) asg(a,1) sat(b) sat(c) sat(a)
SATISFIABLE
```

The encoding π_{adm} gives us an encoding of the enumeration (as well as existence) problem for the admissible semantics. Skeptical reasoning for the admissible semantics is trivial (as the interpretation mapping every statement to **u** is always admissible), but note that via credulous reasoning for ASP programs we directly obtain an encoding for credulous acceptance w.r.t. the admissible semantics from π_{adm}. Moreover, the encoding is adequate from the point of view of the complexity as $\pi_{adm}(D)$ is a normal logic program for any ADF D. Also, given our recursive definition of the evaluation of the acceptance conditions within ASP rules, the arity of predicates in our encodings are bounded (in fact, the maximum arity of predicates is of size two). Credulous reasoning for normal logic programs with predicates of bounded arity just as credulous acceptance w.r.t the admissible semantics is Σ_2^P-complete.

3.2.2 Encodings for the complete semantics

For the ASP encoding of the complete semantics we only need to add two constraints to the encoding of the admissible semantics. These express a further condition that an interpretation v for an ADF $D = (S, \{\phi_s\}_{s \in S})$ has to fulfill to be complete, in addition to not having counter-models for being an admissible interpretation as expressed in Section 3.2.1. The condition in question is that for every $s \in S$:

- if $v(s) = \mathbf{u}$ then there are $w_1, w_2 \in [v]_2$ s.t. $w_1(\phi_s) = 0$ and $w_2(\phi_s) = 1$

Expressing this condition in the form of constraints gives us the encoding

$$\begin{aligned}\pi_{com}(D) :=&\pi_{adm}(D) \cup \\ &\{ :\!-\, arg(S), asg(S, \mathbf{u}), not\ inv(S). \\ &\ \ :\!-\, arg(S), asg(S, \mathbf{u}), not\ sat(S).\}\end{aligned}$$

Proposition 11. *For every ADF D it holds that $com(D) \cong \mathcal{AS}(\pi_{com}(D))$.*

Proof. (sketch) The proof is similar to that of Proposition 10. □

The encoding $\pi_{com}(D)$ is of the enumeration (and existence) problem w.r.t. the complete semantics. Note that since credulous acceptance for the complete semantics is equivalent to credulous acceptance for the admissible semantics, we obtain complexity adequate encodings for the complete semantics via the encoding presented in Section 3.2.1. Skeptical acceptance, on the other hand, is equivalent to skeptical (= credulous) acceptance for the grounded semantics; we therefore obtain encodings via the encoding for the grounded semantics presented in sections 3.2.4 and 3.2.6.

3.2.3 Encodings for the preferred semantics

For the encoding of the preferred semantics we make use of the saturation technique [EG95]; see [CDG$^+$15] for its use in computing the preferred extensions of Dung AFs. The saturation technique allows checking that a property holds for a *set* of guesses within a disjunctive ASP program, by generating a unique "saturated" guess that "verifies" the property for any such guess. Existence of a non-saturated guess hence implies that the property of interest does not hold for the guess in question.

In the encoding of the preferred semantics for an ADF D we extend $\pi_{adm}(D)$ by making use of the saturation technique to verify that all interpretations of D that are greater w.r.t. $\leq_i$ than the interpretation determined by the assignments guessed in the program fragment π_{guess} are either identical to the interpretation in question or not admissible. As a consequence, the relevant interpretation must be preferred according to the definition of this semantics for ADFs.

The module π_{guess2} amounts to "making a second guess" (indicated by the predicate *asg2*) extending the "first guess" (*asg*) from π_{guess}.

$$\begin{aligned}\pi_{\text{guess2}} := \{&\mathit{asg2}(S,0) \mathbin{:-} \mathit{asg}(S,0).\\ &\mathit{asg2}(S,1) \mathbin{:-} \mathit{asg}(S,1).\\ &\mathit{asg2}(S,1) \vee \mathit{asg2}(S,0) \vee \mathit{asg2}(S,\mathbf{u}) \mathbin{:-} \mathit{asg}(S,\mathbf{u}).\}\end{aligned}$$

The fragment $\pi_{\mathit{sat2}}(D)$ will allow us to check whether the second guess obtained from π_{guess2} is admissible:

$$\begin{aligned}\pi_{\mathit{sat2}}(D) := \{&\mathit{sat2}(s) \mathbin{:-} \omega 2_s, V_{\phi_s} = 1.\\ &\mathit{inv2}(s) \mathbin{:-} \omega 2_s, V_{\phi_s} = 0. \mid s \in S\}\end{aligned}$$

with

$$\omega 2_s := \{\mathit{asg2}(t, Y_t), \mathit{lt}(Y_t, V_t) \mid t \in \mathit{par}_D(s)\} \cup \Omega(\phi_s).$$

The only difference between the fragment $\pi_{\mathit{sat2}}(D)$ and $\pi_{\mathit{sat}}(D)$ is that we now evaluate acceptance conditions w.r.t. completions of the second guess given via the predicate *asg2*.

The following program fragment guarantees that the atom *saturate* is derived whenever the second guess (computed via π_{guess2}) is either identical (first rule of $\pi_{\mathit{check}}(D)$) to the first guess (computed via the module π_{guess}) or is not admissible (last two rules of $\pi_{\mathit{check}}(D)$). We will say that in this case the second guess is *not* a counter-example to the first guess corresponding to a preferred interpretation of D. We here assume that the statements S of D are numbered, i.e. $S = \{s_1, \ldots, s_k\}$.

$$\begin{aligned}\pi_{\mathit{check}}(D) := \{&\mathit{saturate} \mathbin{:-} \mathit{asg}(s_1, X_1), \mathit{asg2}(s_1, X_1), \ldots\\ &\qquad\qquad \mathit{asg}(s_k, X_k), \mathit{asg2}(s_k, X_k).\\ &\mathit{saturate} \mathbin{:-} \mathit{asg2}(S,1), \mathit{inv2}(S).\\ &\mathit{saturate} \mathbin{:-} \mathit{asg2}(S,0), \mathit{sat2}(S).\}\end{aligned}$$

The module π_{saturate} now assures that whenever the atom *saturate* is derived, first of all $\mathit{asg2}(s,0)$, $\mathit{asg2}(s,1)$, and $\mathit{asg2}(s,\mathbf{u})$ are derived for every $s \in S$ for which $\mathit{asg}(s,\mathbf{u})$ has been derived. Also, $\mathit{sat2}(s)$ and $\mathit{inv2}(s)$ are derived for every $s \in S$.

$$\pi_{\mathit{saturate}} := \{\mathit{asg2}(S,0) \mathbin{:-} \mathit{asg}(S,\mathbf{u}), \mathit{saturate}.$$

$$\begin{aligned}
&\mathit{asg2}(S, 1) \mathrel{:-} \mathit{asg}(S, \mathbf{u}), \mathit{saturate}.\\
&\mathit{asg2}(S, \mathbf{u}) \mathrel{:-} \mathit{asg}(S, \mathbf{u}), \mathit{saturate}.\\
&\mathit{sat2}(S) \mathrel{:-} \mathit{arg}(S), \mathit{saturate}.\\
&\mathit{inv2}(S) \mathrel{:-} \mathit{arg}(S), \mathit{saturate}.\}
\end{aligned}$$

The effect of this fragment is that whenever all the "second guesses" (computed via π_{guess2}) are *not* counter-examples to the first guess (computed via π_{guess}) corresponding to a preferred interpretation of D, then all the answer sets will be saturated on the predicates *ass2*, *sat2*, and *inv2*, i.e. the same ground instances of these predicates will be included in any answer set. Thus, all answer-sets (corresponding to the ADF interpretation determined by the first guess) will be indistinguishable on the new predicates used for the encoding of the preferred interpretation; meaning: those not in $\pi_{adm}(D)$. On the other hand, were there to be a counter-example to the first guess corresponding to an interpretation of D, then a non-saturated and hence smaller (w.r.t $\subseteq$) answer set could be derived. We disallow the latter by adding to the program fragments $\pi_{adm}(D)$, π_{guess2}, $\pi_{sat2}(D)$, $\pi_{check}(D)$, $\pi_{saturate}$, a constraint filtering out precisely such answer sets. The latter being those for which the atom *saturate* is *not* derived. We thus arrive at the following encoding for the preferred semantics:

$$\begin{aligned}
\pi_{prf}(D) :=& \pi_{adm}(D) \ \cup \ \pi_{guess2} \ \cup \ \pi_{sat2}(D) \ \cup\\
& \pi_{check}(D) \ \cup \ \pi_{saturate} \ \cup \ \{ \mathrel{:-} \mathit{not\ saturate}.\}
\end{aligned}$$

Proposition 12. *For every ADF D it holds that $prf(D) \cong \mathcal{AS}(\pi_{prf}(D))$,*

Proof. Let $D = (S, \{\phi_s\}_{s \in S})$ be an ADF and $v \in prf(D)$. Let also

$$I' := I \cup \{\mathit{asg2}(s, 1) \mid s \in S, v(s) = 1\} \cup \{\mathit{asg2}(s, 0) \mid s \in S, v(s) = 0\} \cup I^{\triangle}$$

be a set of ground atoms where I is defined as in the "only if" direction of the proof of Proposition 10 (hence, $v \cong I'$). Moreover, $I^{\triangle}$ is the set of ground atoms forming the "saturation" of the predicates *asg2*, *sat2*, *inv2*, *saturate* (*asg2* is saturated only for $s \in S$ s.t. $v(s) = \mathbf{u}$) defined as

$$\begin{aligned}
I^{\triangle} :=& \{\mathit{asg2}(s, x) \mid s \in S, x \in \{1, 0, \mathbf{u}\}, v(s) = \mathbf{u}\} \cup\\
& \{\mathit{sat2}(s) \mid s \in S\} \cup\\
& \{\mathit{inv2}(s) \mid s \in S\} \cup\\
& \{\mathit{saturate}\}
\end{aligned}$$

Note first that since none of the predicates occurring in $I' \setminus I$ appear in $\pi_{adm}(D)$, we have that $\pi_{adm}(D)^{I'} = \pi_{adm}(D)^{I}$. As thus also all of the atoms appearing in $\pi_{adm}(D)^{I'}$ that are in I' are those which are in I, we have that I' and I satisfy the bodies and heads of the same rules in $\pi_{adm}(D)^{I'}$. By the proof of the "only if" direction of Proposition 10 (i.e. that I satisfies $\pi_{adm}(D)^{I} = \pi_{adm}(D)^{I'}$) it then follows that I' satisfies $\pi_{adm}(D)^{I'}$.

I' also satisfies each of $\pi_{sat2}(D)^{I'} = Gr(\pi_{sat2}(D))$, $\pi_{check}(D)^{I'} = Gr(\pi_{check}(D))$ as the heads of all possible ground instances of the rules of each of the modules $\pi_{sat2}(D)$ and $\pi_{check}(D)$ is contained in $I^{\triangle} \subset I'$. Moreover, I' satisfies all groundings of the first two rules of π_{guess2} (that are in $\pi_{guess2}^{I'} = Gr(\pi_{guess2})$) as both $asg(s,x) \in I'$ and $asg2(s,x) \in I'$ whenever $v(s) = x$ for $x \in \{1,0\}$. I' also satisfies all groundings of the third rule of π_{guess2} as whenever $asg(s,u) \in I'$ this means that $v(s) = \mathbf{u}$ and then $asg2(s,x) \in I^{\triangle} \subset I'$ for every $x \in \{1,0,\mathbf{u}\}$. For the same reason I' also satisfies all possible groundings of the first three rules of $\pi_{saturate}$ (contained in $\pi_{saturate}^{I'} = Gr(\pi_{saturate})$). Furthermore, I' satisfies all possible groundings of the last two rules of $\pi_{saturate}$ since whenever $arg(s) \in I'$ this means that $s \in S$ and then $sat2(s) \in I^{\triangle} \subset I'$ as well as $inv2(s) \in I^{\triangle} \subset I'$. Finally, since $saturate \in I^{\triangle} \subset I'$ the constraint

$$:\!-not\ saturate.$$

is deleted from $\pi_{prf}(D)$ when forming the reduct $\pi_{prf}(D)^{I'}$. We thus have that I' satisfies all of the rules in $\pi_{prf}(D)^{I'}$; hence, I' satisfies $\pi_{prf}(D)^{I'}$.

Consider now that there is a $I'' \subset I'$ that satisfies $\pi_{prf}(D)^{I'}$. Since I'' satisfies $\pi_{adm}(D)^{I'} = \pi_{adm}(D)^{I}$, we have by the argument in the "only if" direction of the proof of Proposition 10 that $I \subseteq I''$. Note that then $asg(s,x) \in I''$ for every $s \in S$ s.t. $v(s) = x$ for $x \in \{1,0\}$. On the other hand, I'' satisfies the groundings of the first two rules in π_{guess2} (since $\pi_{guess2}^{I'} = Gr(\pi_{guess2})$). It hence follows that also $asg2(s,x) \in I''$ for every $s \in S$ s.t. $v(s) = x$ for $x \in \{1,0\}$. Moreover, since I'' satisfies the groundings of the last rule in π_{guess2} and $\{asg(s,\mathbf{u}) \mid s \in S, v(s) = \mathbf{u}\} \subset I \subset I''$ it must be the case that there is some $x \in \{\mathbf{u},1,0\}$ s.t. $asg2(s,x) \in I''$ for every $s \in S$ s.t. $v(s) = \mathbf{u}$. We thus have that there is an ADF interpretation $v' \geq_i v$ s.t. there is an atom $asg2(s,x) \in I''$ whenever $v'(s) = x$.

Assume now that $saturate \notin I''$. Since I'' satisfies $\pi_{saturate}^{I'} = Gr(\pi_{saturate})$ this means that $B(r) \not\subset I''$ for every $r \in Gr(\pi_{saturate})$. Hence, in particular, $B(r) \not\subset I''$ for the rule r

$$\begin{aligned} saturate\!:\!-asg(s_1, v(s_1)), asg2(s_1, v'(s_1)), \ldots \\ asg(s_k, v(s_k)), asg2(s_k, v'(s_k)). \end{aligned}$$

This amounts to $v \neq v'$ and, hence, $v <_i v'$. Also, $B(r) \not\subset I''$ for the rule r

$$\mathit{saturate} \text{:-} \mathit{asg2}(s,1), \mathit{inv2}(s).$$

for every $s \in S$. This amounts to (since I'' satisfies $\pi_{sat2}(D)^{I'} = Gr(\pi_{sat2}(D))$; see proof of Proposition 10) there not being any $s \in S$ and $w \in [v']_2$ for which $v'(s) = 1$ and $w(s) = 0$. In the same manner the fact that $B(r) \not\subseteq I''$ for the rule r

$$\mathit{saturate} \text{:-} \mathit{asg2}(s,0), \mathit{sat2}(s).$$

for every $s \in S$, means that there is no $s \in S$ and $w \in [v']_2$ for which $v'(s) = 0$ and $w(s) = 1$. But then $v' \in adm(D)$ which, together with the fact that $v <_i v'$, is a contradiction to $v \in prf(D)$.

On the other hand if $\mathit{saturate} \in I''$, since I'' satisfies all possible groundings of the first three rules of $\pi_{saturate}$ (as $\pi^{I'}_{saturate} = Gr(\pi_{saturate})$), it would be the case that whenever $asg(s, \mathbf{u}) \in I''$ and hence $v(s) = \mathbf{u}$ (since $I \subset I''$) also $\mathit{asg2}(s,x) \in I''$ for every $x \in \{\mathbf{u}, 0, 1\}$. Moreover, if $\mathit{saturate} \in I''$, since I'' satisfies all possible groundings of the last two rules of $\pi_{saturate}$, it would also follow that $sat(s) \in I''$ as well as $inv(s) \in I''$ for every $s \in S$. This means that if $\mathit{saturate} \in I''$, then $I' \subseteq I''$. This is a contradiction to our assumption that $I'' \subset I'$. In conclusion, there is no $I'' \subset I'$ that satisfies $\pi_{prf}(D)^{I'}$. Therefore $I' \in \mathcal{AS}(\pi_{prf}(D))$.

We turn now to proving that for any $I \in \mathcal{AS}(\pi_{prf}(D))$ it holds that $v \in prf(D)$ for $v \cong I$. Note first of all that since I satisfies $\pi_{adm}(D)^I$ by the proof of the "if" direction of Proposition 10 we obtain that v is well defined and, moreover, $v \in adm(D)$.

Since $v \in adm(D)$, $v \notin prf(D)$ would mean that there is a $v' \in adm(D)$ s.t. $v' >_i v$. Now, notice first of all that since $I \in \mathcal{AS}(\pi_{prf}(D))$, $\mathit{saturate} \in I$ since otherwise the constraint

$$\text{:-} \mathit{not\ saturate}.$$

would not be deleted from $\pi_{prf}(D)$ (as must be the case) when forming the reduct $\pi_{prf}(D)^I$ and hence $\pi_{prf}(D)$ would have no answer set. We know from the proof of the "only if" direction of Proposition 12 that from $\mathit{saturate} \in I$ it then follows that $I^{\triangle} \subseteq I$ where

$$\begin{aligned} I^{\triangle} = &\{\mathit{asg2}(s,x) \mid s \in S, x \in \{1,0,\mathbf{u}\}, v(s) = \mathbf{u}\} \cup \\ &\{\mathit{sat2}(s) \mid s \in S\} \cup \end{aligned}$$

$$\{\mathit{inv2}(s) \mid s \in S\} \cup$$
$$\{\mathit{saturate}\}.$$

Now let us define

$$\begin{aligned} I' := \cup_{p \in \{arg,lt,asg,sat,inv\}} I_p \cup \{\mathit{asg2}(s, v'(s)) \mid s \in S\} \cup \\ \{\mathit{sat2}(s) \mid s \in S, \text{ there is a } w \in [v']_2 \text{ s.t. } w(\phi_s) = 1\} \cup \\ \{\mathit{inv2}(s) \mid s \in S, \text{ there is a } w \in [v']_2 \text{ s.t. } w(\phi_s) = 1\} \end{aligned}$$

for which by construction (and $v' >_i v$) $I' \subset I$ holds. Notice first of all that since all negative atoms of $\pi_{prf}(D)$ occur in $\pi_{adm}(D) \cup \{\textbf{:-}\mathit{not\ saturate.}\}$ we have that

$$\pi_{prf}(D)^I = \pi_{adm}(D)^I \cup Gr(\pi_{guess2}) \cup Gr(\pi_{sat2}(D)) \cup Gr(\pi_{check}(D)) \cup Gr(\pi_{saturate}).$$

Now, since I and I' are the same when considering the atoms occurring in $\pi_{adm}(D)^I$ ($\cup_{p \in \{arg,lt,asg,sat,inv\}} I_p \subset I'$) and I satisfies $\pi_{adm}(D)^I$ so does I'. Moreover, since $v' >_i v$ by construction $\mathit{asg2}(s,x) \in I'$ whenever $\mathit{asg}(s,x) \in I$ for $x \in \{1,0\}$ and there is a $y \in \{\mathbf{u},1,0\}$ s.t. $\mathit{asg2}(s,y) \in I'$ whenever $\mathit{asg}(s,\mathbf{u}) \in I$. Hence I' also satisfies $\pi^I_{guess2} = Gr(\pi_{guess2})$.

Using analogous arguments as in the "only if" direction of the proof of Proposition 10, from the fact that $\mathit{asg2}(s,x) \in I'$ iff $v'(s) = x$ (for $s \in S$ and $x \in \{1,0,\mathbf{u}\}$) and the definition for when $\mathit{sat2}(s)$ and $\mathit{inv2}(s)$ are in I', it follows that v' satisfies $\pi^I_{sat2} = Gr(\pi_{sat2})$. We have also seen in the proof of the "only if" direction of Proposition 12 that $v' \neq v$ and $v' \in adm(D)$ implies that I' does not satisfy the body of any of the rules in $\pi_{check}(D)^I = Gr(\pi_{check}(D))$. Finally, since $\mathit{saturate} \notin I'$ it is also the case that I' satisfies $\pi^I_{saturate} = Gr(\pi_{saturate})$. In conclusion, we have that I' satisfies $\pi_{prf}(D)^I$ and $I' \subset I$ which contradicts $I \in \mathcal{AS}(\pi_{prf}(D)))$. Hence, there cannot be a $v' >_i v$ s.t. $v' \in adm(D)$ and therefore $v \in prf(D)$ must be the case. □

Example 9. *The encoding $\pi_{prf}(D)$ for the ADF D from Example 1 as implemented by our system* YADF *looks as follows:*

```
leq(u,0).
leq(u,1).
leq(0,0).
leq(1,1).
arg(a).
arg(b).
arg(c).
```

```
asg(S,u) :- arg(S),not asg(S,0),not asg(S,1).
asg(S,0) :- arg(S),not asg(S,1),not asg(S,u).
asg(S,1) :- arg(S),not asg(S,u),not asg(S,0).
sat(a) :- asg(b,Y0),leq(Y0,V0),V1=1-V0,V2=V1?V0,V2=1.
sat(b) :- asg(b,Y0),leq(Y0,V0),V0=1.
sat(c) :- asg(c,Y0),leq(Y0,V0),asg(b,Y1),leq(Y1,V1),
                                  V3=1,V3=V2?V1,V2=1-V0.
inv(a) :- asg(b,Y0),leq(Y0,V0),V1=1-V0,V2=V1?V0,V2=0.
inv(c) :- asg(c,Y0),leq(Y0,V0),asg(b,Y1),leq(Y1,V1),
                                  V3=V2?V1,V3=0,V2=1-V0.
inv(b) :- asg(b,Y0),leq(Y0,V0),V0=0.
:- arg(S),asg(S,1),inv(S).
:- arg(S),asg(S,0),sat(S).
asg2(S,0) :- asg(S,0).
asg2(S,1) :- asg(S,1).
asg2(S,0)|asg2(S,1)|asg2(S,u) :- asg(S,u).
sat2(a) :- asg2(b,Y0),leq(Y0,V0),V1=1-V0,V2=V1?V0,V2=1.
sat2(b) :- asg2(b,Y0),leq(Y0,V0),V0=1.
sat2(c) :- asg2(c,Y0),leq(Y0,V0),asg2(b,Y1),leq(Y1,V1),
                                  V3=1,V3=V2?V1,V2=1-V0.
inv2(b) :- asg2(b,Y0),leq(Y0,V0),V0=0.
inv2(c) :- asg2(c,Y0),leq(Y0,V0),asg2(b,Y1),leq(Y1,V1),
                                  V3=V2?V1,V3=0,V2=1-V0.
inv2(a) :- asg2(b,Y0),leq(Y0,V0),V1=1-V0,V2=V1?V0,V2=0.
saturate :- asg(c,X0),asg2(c,X0),asg(b,X1),asg2(b,X1),
                                  asg(a,X2),asg2(a,X2).
saturate :- arg(S),asg2(S,0),sat2(S).
saturate :- arg(S),asg2(S,1),inv2(S).
asg2(S,u) :- asg(S,u),saturate.
asg2(S,0) :- asg(S,u),saturate.
asg2(S,1) :- asg(S,u),saturate.
sat2(S) :- arg(S),saturate.
inv2(S) :- arg(S),saturate.
:- not saturate.
```

An output of an ASP solver given this instance looks as follows (only showing asg and saturate predicates):

```
Answer: 1
asg(c,u) asg(b,0) asg(a,1) saturate
Answer: 2
asg(c,1) asg(b,1) asg(a,1) saturate
SATISFIABLE
```

Note that π_{prf}, in addition to providing an encoding of the enumeration problem for the preferred semantics, gives us adequate encodings from the complexity point of view for skeptical acceptance. The latter via skeptical acceptance of ASP disjunctive programs with predicates of bounded arity. Existence and credulous acceptance for the preferred semantics are equivalent to existence and credulous acceptance for the admissible semantics; the encodings given in section 3.2.1 (and 3.2.6) hence provide adequate encodings for these reasoning tasks.

3.2.4 Encodings for the grounded semantics

Our encoding for the grounded semantics is based on the fact that (see [SW15]) $v \in grd(D)$ for an interpretation v and an ADF $D = (S, \{\phi_s\}_{s \in S})$ if v is the $\leq_i$-minimal interpretation satisfying

- for each $s \in S$ such that $v(s) = 1$ there exists an interpretation $w \in [v]_2$ for which $w(\phi_s) = 1$,
- for each $s \in S$ such that $v(s) = 0$ there exists an interpretation $w \in [v]_2$ for which $w(\phi_s) = 0$, and
- for each $s \in S$ such that $v(s) = \mathbf{u}$ there exist interpretations $w_1 \in [v]_2$ and $w_2 \in [v]_2$ such that $w_1(\phi_s) = 1$ and $w_2(\phi_s) = 0$.

We say that an interpretation v for the ADF D that satisfies one of the above for $s \in S$, that it satisfies the properties for being a candidate for being the grounded interpretation w.r.t. s. The completions w_1 and w_2 verify this fact for v and s. If v satisfies the properties w.r.t. every $s \in S$ then v is a candidate for being the grounded interpretation of D. An interpretation $v' <_i v$ that is also a candidate for being the grounded interpretation is a counter-model (alternatively, counter-example) to v being the right candidate (for being the grounded interpretation).

Our encoding for the grounded semantics essentially consists first of all, once more in the guessing part π_{guess} where we guess assignments of truth values to the statements of the ADF of interest D. This corresponds to guessing an interpretation v for D. Constraints in our encoding filter out guessed interpretations which either are not candidates to being the grounded interpretation or which have counter-models to being the right candidate. These constraints rely on the rules in $\pi_{sat}(D)$ and rules defining when an interpretation has a counter-model to being the right candidate respectively.

We start of with a few facts needed for our encoding. First of all, we use different facts to those in π_{lt} for encoding the truth values a possible counter-model to the interpretation guessed via π_{guess} (being the right candidate for the grounded interpretation) can take. The reason is we need an additional argument (the first argument of the predicate *lne*) allowing us to check whether the interpretation in question is distinct to the one determined by the predicate *asg*.

$$\begin{aligned}\pi_{lne} :=& \{ lne(1,\mathbf{u},1).\ lne(1,\mathbf{u},0).\} \cup \\ & \{ lne(0,1,1).\ lne(0,0,0).\ lne(0,\mathbf{u},\mathbf{u}).\}\end{aligned}$$

Secondly, we need a set of facts for checking whether an interpretation satisfies the properties required for candidates to being the grounded interpretation. Specifically, given a statement s of the ADF D, $prop(x, y, z)$ can be used to check whether the correct relationship between $v(s)$, $w_1(s)$, and $w_2(s)$ holds for an interpretation v for D, and $w_1, w_2 \in [v]_2$. In particular, note that $w_1 = w_2$ is possible and hence $prop(x, y, z)$ can also be used to check the properties for when $v(s) = x$ and $x \in \{1, 0\}$.

$$\pi_{prop} := \{ prop(1,1,1).\ prop(0,0,0).\ prop(\mathbf{u},0,1).\}$$

The module consisting of constraints checking whether the interpretation corresponding to the assignments guessed via π_{guess} is a candidate for being the grounded interpretation assumes, as we stated earlier, that the rules in $\pi_{sat}(D)$ are also a part of the encoding for the grounded semantics.

$$\begin{aligned}\pi_{ca}(D) :=& \{ {:-}\, arg(S), asg(S,1), not\ sat(S).\ \ {:-}\, arg(S), asg(S,0), not\ inv(S).\} \cup \\ & \{ {:-}\, arg(S), asg(S,\mathbf{u}), not\ inv(S).\ \ {:-}\, arg(S), asg(S,\mathbf{u}), not\ sat(S).\}\end{aligned}$$

Now, note that the grounded interpretation can be obtained via choosing the minimal interpretation w.r.t. $\leq_i$ from all interpretations that correspond to some answer set of the encoding

$$\pi_{ca\text{-}grd}(D) := \pi_{arg}(D) \cup \pi_{lne} \cup \pi_{prop} \cup \pi_{guess} \cup \pi_{ca}(D);$$

i.e. what essentially boils down to a skeptical acceptance problem for $\pi_{ca\text{-}grd}(D)$.

In order to obtain an encoding not requiring processing of all answer sets we need a rule defining when an interpretation is a counter-model to the interpretation determined via the predicate *asg* being the right candidate. For this we will need to make repeated use of the function Ω within a single rule and therefore first of all make the symbol ranging over the ASP-variables representing subformulas of a propositional formula ϕ within $\Omega(\phi)$ an explicit parameter of the function. This is straightforward, but for completeness we give the full definition of our parametrised version of the function Ω. Here ϕ is once more a propositional formula built from propositional variables in a set P, while now V is an arbitrary (meta-) symbol used to refer to the variables introduced by the function:

$$\Omega^V(\phi) := \begin{cases} \Omega^V(\phi_1) \cup \Omega^V(\phi_2) \cup \{V_\phi = V_{\phi_1} \& V_{\phi_2}\} & \text{if } \phi = \phi_1 \wedge \phi_2 \\ \Omega^V(\phi_1) \cup \Omega^V(\phi_2) \cup \{V_\phi = V_{\phi_1} ? V_{\phi_2}\} & \text{if } \phi = \phi_1 \vee \phi_2 \\ \Omega^V(\psi) \cup \{V_\phi = 1 - V_\psi\} & \text{if } \phi = \neg\psi \\ \emptyset & \text{if } \phi = p \in P \end{cases}$$

Again, V_ϕ, V_{ϕ_1} V_{ϕ_2} and V_ψ are variables representing the subformulas of ϕ. From now on, whenever we introduce sets $\Omega^{V_1}(\phi_1)$ and $\Omega^{V_2}(\phi_2)$ for possibly identical formulas ϕ_1 and ϕ_2 but distinct symbols V_1 and V_2, we implicitly also assume that that then $\Omega^{V_1}(\phi_1) \cap \Omega^{V_2}(\phi_2) = \emptyset$.

Our rule for defining counter-models to an interpretation being the right candidate for the grounded interpretation requires first of all a part for "generating" an interpretation less informative (w.r.t $\leq_i$) and distinct than the interpretation determined by π_{guess}; i.e. a candidate counter-model. For this we use the atoms

$$\lambda_D := \{asg(s, X_s), lne(E_s, Y_s, X_s) \mid s \in S\} \cup \Omega^E(\vee_{s \in S} s) \cup \{E_{\vee_{s \in S} s} = 1\}.$$

On the other hand, we introduce the following set of atoms to check whether the candidate counter-model is indeed a counter-model to the interpretation determined by *asg* being the right candidate. We need to check the properties candidates for being the grounded interpretation need to satisfy for each of the statements s of the ADF of interest D; therefore the need for having sets of atoms $\kappa_{s,D}$ defined for every statement s:

$$\begin{aligned} \kappa_{s,D} := &\{lt(Y_t, V^{(t,s),1}) \mid t \in par_D(s)\} \cup \Omega^{V^{(t,s),1}}(\phi_s) \cup \\ &\{lt(Y_t, V^{(t,s),2}) \mid t \in par_D(s)\} \cup \Omega^{V^{(t,s),2}}(\phi_s) \cup \\ &\{prop(Y_s, V^{(t,s),1}_{\phi_s}, V^{(t,s),2}_{\phi_s})\}\} \end{aligned}$$

Note here the use of the predicate *lt* (rather than *lne*) and hence the need for also having the module π_{lt} within the encoding for the grounded semantics. Putting all the above together we have a quite large rule defining when a candidate counter-model is indeed a counter-model to the interpretation given by π_{guess} being the right candidate for the grounded interpretation:

$$\pi_{cm}(D) := \{cm\colon\!- \lambda_D \cup \{\kappa_{s,D} \mid s \in S\}.\}$$

The following is then an encoding for the grounded interpretation for the ADF D; allowing to compute this interpretation in one go:

$$\pi_{grd}(D) := \pi_{arg}(D) \cup \pi_{lt} \cup \pi_{lne} \cup \pi_{prop} \cup \pi_{guess} \cup \pi_{ca}(D) \cup \pi_{cm}(D) \cup \{\colon\!- cm.\}$$

Proposition 13. *For every ADF D it holds that $grd(D) \cong \mathcal{AS}(\pi_{grd}(D))$,*

Proof. The proof is similar to that of Proposition 14. □

We do not obtain complexity sensitive encodings for credulous and skeptical reasoning w.r.t. the grounded semantics via the encoding π_{grd}. Nevertheless, the encoding offers an alternative strategy to obtaining the (unique) grounded interpretation of an ADF to that of the static encodings at the basis of the `DIAMOND` family of systems mentioned in the introduction to this work (Section 1). Also, the encoding forms the basis of the complexity adequate encoding for the stable semantics we present in Section 3.2.5.

3.2.5 Encodings for the stable semantics

As already hinted at and is to be expected, given the definition of this semantics (Definition 9), our encoding for the stable semantics is based on the encoding for the grounded semantics. Nevertheless, some modifications are required. First of all we need to guess assignments to statements of an ADF D corresponding to a two valued rather than a three valued interpretation v for D. Secondly, we need to check that v is a model of D. Third, we need to ensure that v assigns the truth value 1 to the same statements as the grounded interpretation of the reduct of D^v (i.e. $v\downharpoonright_{E_v} = grd(D^v)$) rather than D simpliciter.

To start we slightly modify some facts used in previous encodings. Our encoding once more follows the guess & check methodology and there will therefore be a part used to guess a candidate v for being the stable interpretation of our ADF of interest D. We will then need a modified version of π_{lt} to set the right truth values for completions of a candidate counter-model v' to v (actually, $v\downharpoonright_{E_v}$) being the right candidate for the grounded interpretation (as explained in Section 3.2.4) of the reduct D^v. Such completions will therefore assign the truth value 0 to any statement s of D whenever $v(s) = 0$ independently of the value of $v'(s)$ (the first argument of the predicate *lt2* is used to refer to the value of $v(s)$).

$$\begin{aligned}\pi'_{lt} :=& \{lt2(1,\mathbf{u},0).\ lt2(1,\mathbf{u},1).\ lt2(1,0,0).\ lt2(1,1,1).\} \cup \\ & \{lt2(0,\mathbf{u},0).\ lt2(0,0,0).\ lt2(0,1,0).\}\end{aligned}$$

The module π_{lne} also needs to be modified (by one fact) to account for the fact that a counter-model v' to $v\downharpoonright_{E_v}$ (where v is the interpretation guessed to be stable) must be distinct on the statements assigned the truth value 1 (i.e. there must be at least one statement s to which v assigns the truth value 1 and v' the truth value $\mathbf{u}$).

$$\pi'_{lne} := \{lne(1,\mathbf{u},1).\ lne(0,\mathbf{u},0).\} \cup$$

$$\{lne(0,1,1).\ lne(0,0,0).\ lne(0,\mathbf{u},\mathbf{u}).\}$$

We also need to modify π_{prop} adding an extra-argument (again, the first one) to indicate whether the property required per statement of an ADF for candidates for the grounded interpretation is verified or not.

$$\begin{aligned}\pi'_{prop} :=&\{prop2(1,1,1,1).\ prop2(1,0,0,0).\ prop2(1,\mathbf{u},0,1).\ prop2(1,\mathbf{u},1,0).\} \cup\\ &\{prop2(0,1,0,1).\ prop2(0,1,1,0).\ prop2(0,1,0,0).\} \cup\\ &\{prop2(0,0,0,1).\ prop2(0,0,1,0).\ prop2(0,0,1,1).\} \cup\\ &\{prop2(0,\mathbf{u},0,0).\ prop2(0,\mathbf{u},1,1).\}\end{aligned}$$

As already indicated, also our encoding for the stable semantics builds on a module guessing possible assignments to the statements of the ADF D. We only need to slightly modify π_{guess} to obtain a conjecture for the stable interpretation corresponding to a two valued rather than three valued interpretation for D:

$$\begin{aligned}\pi'_{guess} := \{&asg(S,0) \mathbin{:-} not\ asg(S,1), arg(S).\\ &asg(S,1) \mathbin{:-} not\ asg(S,0), arg(S).\}\end{aligned}$$

In order to check that the guessed interpretation is a model of D we again need to evaluate the acceptance conditions of D but this time by the guessed interpretation. For this we make use of the following sets of atoms per statement s of D:

$$\mu_s := \{asg(t, V_t) \mid t \in par_D(s)\} \cup \Omega(\phi_s).$$

The following are then constraints, one per statement, filtering out guesses that are not models of D:

$$\pi_{model}(D) := \{\mathbin{:-}\ asg(s, V_s), \mu_s, V_s \neq V_{\phi_s}.\mid s \in S\}.$$

Now, note that for any $v \in mod(D)$, $v \downharpoonright_{E_v}$ is a candidate to being the grounded interpretation of the reduct D^v. The reason is first of all that $[v \downharpoonright_{E_v}]_2 = \{v \downharpoonright_{E_v}\}$ and, hence, for any $w \in [v \downharpoonright_{E_v}]_2$, $w(\phi'_s) = v \downharpoonright_{E_v} (\phi'_s) = v(\phi_s)$ for the modified acceptance conditions $\phi'_s = \phi_s[b/\bot : v(b) = 0]$ of D^v. As a consequence, clearly whenever $v \downharpoonright_{E_v} (s) = x$ for $x \in \{1, 0\}$ (in fact, $x = 1$) there is a $w \in [v]_2$, namely $w = v \downharpoonright_{E_v}$, for which $w(\phi_s) = x$. In effect, the latter is the case by virtue of $v \in mod(D)$ and hence

$v(\phi_s) = v \downharpoonright_{E_v} (\phi'_s) = x$ whenever $v(s) = x$. Also, there are no statements for which $v \downharpoonright_{E_v} (s) = \mathbf{u}$. The consequence for our encoding for the stable semantics is that $\pi_{model}(D)$ suffices for checking whether our guessed interpretation, when projected on the statements to which it assigns the truth value 1, is a candidate for being the grounded interpretation of D^v.

All that remains for our encoding of the stable semantics is therefore, as we have for the encoding of the grounded semantics, a constraint filtering out guessed interpretations which have counter-models to being the right candidate for being the grounded interpretation of D^v. For this we introduce a slightly modified version of $\pi_{cm}(D)$ accounting for the fact that we need to check for counter-models to $v \downharpoonright_{E_v}$ being the right candidate for the reduct D^v rather than v and D. This means that completions of potential counter-models need to set any statement set to the truth value 0 by v also to 0. To encode this we use the predicate *lt2* rather than *lt* in our modified version $\kappa'_{s,D}$ of the set of atoms $\kappa_{s,D}$. Also, we need to check the properties that candidates of the grounded interpretation need to satisfy only for statements s for which $v(s) = 1$. To encode this we make use of the predicate *prop2* rather than *prop* and add a corresponding check using ASP built in boolean arithmetic functions.

$$\begin{aligned}\kappa'_{s,D} :=& \{\mathit{lt2}(X_t, Y_t, V^{(t,s),1}) \mid t \in \mathit{par}_D(s)\} \cup \Omega^{V^{(t,s),1}}(\phi_s) \cup \\ & \{\mathit{lt2}(X_t, Y_t, V^{(t,s),2}) \mid t \in \mathit{par}_D(s)\} \cup \Omega^{V^{(t,s),2}}(\phi_s) \cup \\ & \{\mathit{prop2}(P_s, Y_s, V^{(t,s),1}_{\phi_s}, V^{(t,s),2}_{\phi_s})\} \cup \{CX_s = 1 - X_s, O_s = P_s?CX_s, O_s = 1\}\end{aligned}$$

Our modified module $\pi'_{cm}(D)$ of $\pi_{cm}(D)$ is then as follows:

$$\pi'_{cm}(D) := \{\mathit{cm}\colon\!-\lambda_D \cup \{\kappa'_{s,D} \mid s \in S\}.\}$$

Note that we here make use of the set of atoms λ_D as defined in Section 3.2.4, yet relying on the definition of the predicate *lne* as given by the module π'_{lne} rather than π_{lne}. Putting everything together the encoding for the stable semantics has the following form:

$$\begin{aligned}\pi_{stb}(D) :=& \pi_{arg}(D) \ \cup\ \pi'_{lt} \ \cup\ \pi'_{lne} \ \cup\ \pi'_{prop} \ \cup \\ & \pi'_{guess} \ \cup\ \pi_{model}(D) \ \cup\ \pi'_{cm}(D) \ \cup\ \{\colon\!-\mathit{cm}.\}\end{aligned}$$

Proposition 14. *For every ADF D it holds that $stb(D) \cong \mathcal{AS}(\pi_{stb}(D))$.*

Proof. Let $D = (S, \{\phi_s\}_{s \in S})$ be an ADF and $v \in stb(D)$. Let also

$$I := \pi_{arg}(D) \ \cup\ \pi'_{lt} \ \cup\ \pi'_{lne} \ \cup\ \pi'_{prop} \ \cup\ \{asg(s,x) \mid s \in S, v(s) = x\}$$

be a set of ground atoms (such that $v \cong I$). (We slightly abuse the notation here by using e.g. $\pi_{arg}(D)$ to refer to the set of atoms rather than the facts in the module.) We prove now that $I \in \mathcal{AS}(\pi_{stb}(D))$.

We start by proving that I satisfies $\pi_{stb}(D)^I$. Note first that I satisfies each of $\pi_{arg}(D)^I = \pi_{arg}(D)$, $\pi'^I_{lt} = \pi'_{lt}$, $\pi'^I_{lne} = \pi'_{lne}$, $\pi'^I_{prop} = \pi'_{prop}$ since all of the facts in each of these modules are in I. I also satisfies

$$\pi'^I_{guess} = \{asg(s,x) \mathbin{:-} arg(s). \mid s \in S, asg(s,y) \notin I, x \in \{1,0\}, y \in (\{1,0\}\ \{x\})\}$$

by the fact that $v \cong I$.

Assume now that I satisfies the body of some constraint in $\pi_{model}(D)^I$, i.e. there is a $s \in S$ and a substitution σ s.t. $asg(s,\sigma(V_s)) \in I$, $asg(t,\sigma(V_t)) \in I$ for each $t \in par_D(s)$, $\sigma(\Omega(\phi_s)) \in I$, and $\sigma(V_s \neq V_{\phi_s}) \in I$. This translates to $v(s) \neq v(\phi_s)$ which means $v \notin mod(D)$ and contradicts $v \in stb(D)$. Therefore I does not satisfy any of the constraints in $\pi_{model}(D)^I$.

Consider on the other hand that I satisfies the body of some rule in $\pi'_{cm}(D)^I$. This means that there is a substitution σ such that first of all $asg(s,\sigma(X_s))) \in I$ as well as $lne(\sigma(E_s),\sigma(Y_s),\sigma(X_s)) \in I$ for every $s \in S$. Also $\sigma(\Omega^E(\vee_{s \in S} s)) \in I$ and $\sigma(E_{\vee_{s \in S} s} = 1) \in I$. All of this together means that $v'(s) <_i v(s)$ for the interpretation v' defined as $v'(s) := \sigma(Y_s)$. Moreover, since I satisfies π'_{lt}, there is an $s \in S$ s.t. $v(s) = 1$ and $v'(s) = \mathbf{u}$. This means that also $E_v \neq \emptyset$ and $v' \downharpoonright_{E_v} (s) <_i v \downharpoonright_{E_v} (s)$ where $v \downharpoonright_{E_v}$ and $v' \downharpoonright_{E_v}$ are interpretations of the reduct D^v.

Secondly, for every $s \in S$ we have that $lt2(\sigma(X_t),\sigma(Y_t),\sigma(V^{(t,s),1})) \in I$ and also $lt2(\sigma(X_t),\sigma(Y_t),\sigma(V^{(t,s),2})) \in I$ for every $t \in par_D(s)$. Consider the interpretations w_i for $i \in \{1,2\}$ defined as $w_i(t) = \sigma(V^{(t,s),i})$ for every $t \in par_D(s)$. Then $w_i(t) \geq_i v'(t)$ whenever $v(t) = 1$, but $w_i(t) = v(t) = 0$ if $v(t) = 0$. This means that $w_i(\phi_s) = w_i \downharpoonright_{E_v} (\phi'_s)$ for $\phi'_s = \phi_s[b/\bot : v(b) = 0]$, and $w_i \downharpoonright_{E_v} \in [v' \downharpoonright_{E_v}]_2$. Now from $\sigma(\Omega^{V^{(t,s),1}}(\phi_s)) \in I$ for every $t \in par_D(s)$, $\sigma(\Omega^{V^{(t,s),2}}(\phi_s)) \in I$ for every $t \in par_D(s)$, and the fact that $prop2(\sigma(P_s),\sigma(Y_s),\sigma(V^{(t,s),1}_{\phi_s}),\sigma(V^{(t,s),2}_{\phi_s})) \in I$ we have that $\sigma(P_s) = 1$ whenever $w_1 \downharpoonright_{E_v}$ and $w_2 \downharpoonright_{E_v}$ verify that $v' \downharpoonright_{E_v}$ satisfies the properties for being a candidate for the grounded interpretation of D^v w.r.t. $s \in E_v$. Otherwise $\sigma(P_s) = 0$. Moreover, from $\sigma(CX_s = 1 - X_s) \in I$, $\sigma(O_s = P_s?CX_s) \in I$, and $\sigma(O_s = 1) \in I$ it follows that either $\sigma(P_s) = 1$ or $\sigma(X_s) = v(s) = 0$ for every $s \in S$.

In other words, whenever $s \in E_v$ (remember: $E_v \neq \emptyset$) there are completions that verify that $v' \downharpoonright_{E_v}$ satisfies the properties for being a candidate for the grounded interpretation of D^v w.r.t. s. This means that $v' \downharpoonright_{E_v}$ is a counter-model to $v \downharpoonright_{E_v}$ being the grounded interpretation of D^v. This is a contradiction to $v \in stb(D)$. Therefore, I does not satisfy the body of any rule in $\pi'_{cm}(D)^I$ and, hence, satisfies $\pi'_{cm}(D)^I$. Finally, since $cm \notin I$, I does not satisfy the body of the constraint $\{\mathbin{:-} cm.\} \in \pi_{stb}(D)^I$. In conclusion, I satisfies $\pi_{stb}(D)^I$.

Now consider any other I' that satisfies $\pi_{stb}(D)^I$. Clearly, since I' satisfies all of the facts in $\pi_{stb}(D)^I$, we have that $\pi_{arg}(D) \cup \pi'_{lt} \cup \pi'_{lne} \cup \pi'_{prop} \subseteq I'$. But also because of the form of π'^{I}_{guess} (see above) and the fact that I' satisfies $\pi_{arg}(D)^I$ it must be the case that $\{asg(s,x) \mid s \in S, v(s) = x\} \subset I'$. This means that in addition to I satisfying $\pi_{stb}(D)^I$ there is also no $I' \subset I$ that satisfies $\pi_{stb}(D)^I$; i.e. we have that $I \in \mathcal{AS}(\pi_{stb}(D))$.

We now turn to proving that for any $I \in \mathcal{AS}(\pi_{stb}(D))$ it holds that $v \in stb(D)$ for $v \cong I$. Note first that for any such I, since I satisfies $\pi_{arg}(D)^I = \pi_{arg}(D)$ and π'^{I}_{guess}, v s.t. $v \cong I$ is well defined. Now assume that $v \notin stb(D)$. Then either i) $v \notin mod(D)$ or ii) $v \in mod(D)$ but $v \downharpoonright_{E_v} \notin grd(D^v)$.

In the first case i) there must be a $s \in S$ s.t. $v(s) \neq v(\phi_s)$. Consider hence the substitution σ defined as $\sigma(V_s) = v(s)$ and $\sigma(V_t) = v(t)$ for $t \in par_D(s)$. This substitution is s.t. $\sigma(B(r)) \subseteq I$ for the constraint in $\pi_{model}(D)$ corresponding to s. This would mean that I does not satisfy $\pi_{stb}(D)^I$ which is a contradiction. Therefore $v \in mod(D)$ which also means that $v \downharpoonright_{E_v}$ is a candidate for the grounded interpretation of D^v (as we argued in detail while explaining our encoding $\pi_{stb}(D)$).

Consider now the case ii). Since $v \downharpoonright_{E_v}$ is a candidate for the grounded interpretation of D^v but $v \downharpoonright_{E_v} \notin grd(D^v)$ this means there is a counter-model $v' \downharpoonright_{E_v}$ for $v \downharpoonright_{E_v}$ being the right candidate for the grounded interpretation of D^v. First define v' to be s.t. $v'(s) = v' \downharpoonright_{E_v} (s)$ for $s \in E_v$ while $v'(s) = v(s)$ for $s \notin E_v$. Define then the substitution σ for which $\sigma(X_s) = v(s)$ and $\sigma(Y_s) = v'(s)$ for every $s \in S$. Since $v' \neq v$ (because $v' \downharpoonright_{E_v} \neq v \downharpoonright_{E_v}$) there must be an $s \in E_v \subseteq S$ for which $v(s) \neq v'(s)$. Set $\sigma(E_s) = 1$ for all such $s \in S$, but $\sigma(E_s) = 0$ whenever $v(s) = v'(s)$. We thus have that $\sigma(\Omega^E(\vee_{s \in S} s)) \in I$ and $E_{\vee_{s \in S} s} = 1 \in I$. Hence, $\sigma(\lambda_D) \in I$.

Now, since $v' \downharpoonright_{E_v}$ is a counter-model to $v \downharpoonright_{E_v}$ being the right candidate for the grounded interpretation of D^v we have that for every $s \in E_v$ there are completions $w_{s,1}$ and $w_{s,2}$ of $v' \downharpoonright_{E_v}$ that are witnesses for $v' \downharpoonright_{E_v}$ satisfying the properties candidates for the grounded interpretation need to satisfy w.r.t. s. Hence, we continue defining the substitution σ s.t. $\sigma(V^{(t,s),i}) = w_{s,i}(t)$ for every $s \in E_v$ and $t \in par_D(s) \cap E_v$. On the other hand $\sigma(V^{(t,s),i}) = 0$ for $t \in par_D(s) \setminus E_v$. Also, $\sigma(P_s) = 1$ for every $s \in E_v$.

For $s \notin E_v$ we on the other hand define $\sigma(V^{(t,s),i}) = w(t)$ $(i \in \{1,2\})$, $t \in par_D(s) \cap E_v$ for some arbitrary $w \in [v']_2$. On the other hand $\sigma(V^{(t,s),i}) = 0$ for $t \in par_D(s) \setminus E_v$. Also, $\sigma(P_s) = 1$ whenever $v'(s) = w(\phi'_s)$ $(\phi'_s = \phi_s[b/\bot : v(b) = 0])$ and $\sigma(P_s) = 0$ otherwise. Then we have that $\sigma(\Omega^{V^{(t,s),i}}(\phi_s)) \in I$ for $s \in S$, $t \in par_D(s)$, $(i \in \{1,2\})$. Also, $\sigma(CX_s = 1 - X_s) \in I$, $\sigma(O_s = P_s?CX_s) \in I$, and $O_s = 1 \in I$ for every $s \in S$ ($\sigma(P_s) = 1$ for $s \in E_v$, while $\sigma(CX_s) = 1$ for $s \notin E_v$). I.e. $\sigma(\kappa'_{s,D}) \in I$ for every $s \in S$. Hence, since also $\sigma(\lambda_D) \in I$, we have that the body of a rule in $\pi'_{cm}(D)^I$ is satisfied by I and, therefore, $cm \in I$. This means that the constraint $:- cm \in \pi_{stb}(D)^I$ is satisfied by I which contradicts $I \in \mathcal{AS}(\pi_{stb}(D))$. Therefore, the case ii) is also not possible and $v \in stb(D)$ must be the case. □

Example 10. *The encoding $\pi_{stb}(D)$ for the ADF D from Example 1 as implemented* *by our system* `YADF` *(we slightly condense the encoding by generating some facts using rules) looks as follows:*

```
arg(a).
arg(b).
arg(c).
val(u).
val(0).
val(1).
lt2(1,u,1).
lt2(1,u,0).
lt2(1,0,0).
lt2(1,1,1).
lt2(0,X,0) :- val(X).
lne(1,u,1).
lne(0,u,0).
lne(0,X,X):- val(X).
prop(1,1,1,1).
prop(1,0,0,0).
prop(1,u,1,0).
prop(1,u,0,1).
prop(0,X1,X2,X3) :- val(X1),val(X2),val(X3),not prop(1,X1,X2,X3).
asg(S,1) :- arg(S),not asg(S,0).
asg(S,0) :- arg(S),not asg(S,1).
:- asg(b,V0),V0!=V0.
:- asg(a,V1),asg(b,V0),V3=V2?V0,V3!=V1,V2=1-V0.
:- asg(c,V0),asg(b,V1),V3=V2?V1,V3!=V0,V2=1-V0.
cm :- asg(c,X0),asg(b,X1),asg(a,X2),lne(E0,Y0,XO),lne(E1,Y1,X1),
      lne(E2,Y2,X2),E20=E0?E1,E21=E2?E20,E21=1,lt2(X0,Y0,V1),
      lt2(X1,Y1,V2),V4=V3?V2,V3=1-V1,lt2(X0,Y0,V5),lt2(X1,Y1,V6),
      V7=1-V5,V8=V7?V6,prop(P0,Y0,V4,V8),CX0=1-X0,OR9=P0?CX0,OR9=1,
      lt2(X1,Y1,V10),lt2(X1,Y1,V11),prop(P1,Y1,V10,V11),CX1=1-X1,
      OR12=P1?CX1,OR12=1,lt2(X1,Y1,V13),V15=V14?V13,V14=1-V13,
      lt2(X1,Y1,V16),V17=1-V16,V18=V17?V16,prop(P2,Y2,V15,V18),
      CX2=1-X2,OR19=P2?CX2,OR19=1.
:- cm.
```

An output of an ASP solver given this instance looks as follows:

```
UNSATISFIABLE
```

We note that also π_{stb}, in addition to giving us an encoding of the enumeration problem for the stable semantics, provides us with complexity sensitive encodings for credulous as well as skeptical reasoning via the corresponding ASP reasoning tasks.

3.2.6 **DIAMOND** style encodings

ASP-based systems for ADFs are best represented by the DIAMOND family of systems[ES14, ES16, SE17]. From the start the systems in the DIAMOND family support reasoning with subclasses of ADFs with the last version of DIAMOND, goDIAMOND[SE17], allowing ADFs with acceptance conditions expressed as propositional formulas and as boolean functions, as well as bipolar ADFs and AFs (previous versions also included prioritized ADFs). The encoding for the different semantics usually stays the same; what changes for the different subclasses are the encodings for the computation of the consequence operators (or characteristic operators) associated to the different formalisms to obtain the semantics.

The different versions of DIAMOND typically rely on static encodings, i.e. the encodings do not change for different ADF instances. This approach is hence limited by the data complexity of ASP (which only reaches the second level of the polynomial hierarchy; see Section 2.2.4). To handle reasoning problems going beyond the second level of the polynomial hierarchy DIAMOND systems transform ADFs with acceptance conditions expressed as propositional formulas to ADFs with boolean formulas; an operation which may involve an exponential blowup.

While the encodings we presented in sections 3.2.1 to 3.2.5 avoid this last problem, an argument can be made for dynamic encodings of the characteristic operator for ADFs; thus keeping in line with the more modular style of the DIAMOND family and also enabling making more or less direct use of the DIAMOND encodings for the different ADF semantics (including semantics as the conflict-free, naive, and stage semantics for which we do not provide encodings in this work). In any case, such alternative encodings are worth having; we therefore give dynamic ASP encodings of the characteristic operator for ADFs using the representation of acceptance conditions as propositional formulas here. The encodings are based on those of goDIAMOND. We give examples of how to tie in this encoding with DIAMOND (specifically, again goDIAMOND) style encodings of some of the semantics and point out some possible advantages of the DIAMOND style encodings over the encodings we presented in sections 3.2.1 to 3.2.5. For clarity we here use the symbol ρ rather than π to distinguish the DIAMOND-style encodings to those presented in the latter sections.

For the encoding of the characteristic operator Γ_D for an ADF $D = (S, \{\phi_s\}_{s \in S})$, we add a further parameter to several modules we have introduced before. The parameter in question, which we represent with the variable I within our encodings, refers to an application or step of the operator Γ_D. In particular we add an argument to our predicate *asg* (the last one) to be able to refer to the number of iterations of Γ_D used to obtain the ADF interpretation represented by ground atoms built using this predicate. A new parametrised version of the module π_{guess} is then obtained as follows:

$$\rho_{guess}(i) := \{asg(S,0,i) \mathbin{:\!-} not\ asg(S,1,i), not\ asg(S,\mathbf{u},i), arg(S).$$

$$\begin{aligned}&asg(S,1,i) \mathbin{:-} not\ asg(S,\mathbf{u},i), not\ asg(S,0,i), arg(S).\\&asg(S,\mathbf{u},i) \mathbin{:-} not\ asg(S,0,i), not\ asg(S,1,i), arg(S).\}\end{aligned}$$

We here assume that the arguments of the ADF D are specified by a separate module defined just as in previous encodings:

$$\rho_{arg}(D) := \pi_{arg}(D) = \{arg(s).\mid s \in S\}$$

We also use parametrised versions of the predicates *sat* and *inv* to refer to the fact that there are completions corresponding to the interpretation obtained via a number of applications of Γ_D of interest that evaluate the acceptance condition of the interpretation in question to the truth value 1 and 0 respectively. For this we first of all define a modified version of the set of atoms ω_s introduced in Section 3.2.1 for a statement s of the ADF D:

$$\omega_s(I) := \{asg(t, Y_t, I), lt(Y_t, V_t) \mid t \in par_D(s)\} \cup \Omega(\phi_s).$$

Our modified version of the module $\pi_{sat}(D)$ is then as follows. Here the predicate *step* is used to refer to the number of iterations or steps the characteristic operator Γ_D has been applied.

$$\begin{aligned}\rho_{sat}(D) := \{&sat(s,I) \mathbin{:-} \omega_s(I), V_{\phi_s} = 1, step(I).\\&inv(s,I) \mathbin{:-} \omega_s(I), V_{\phi_s} = 0, step(I).\mid s \in S\}.\end{aligned}$$

The following module now provides shortcuts for when there, first of all, are completions corresponding to the interpretation obtained via a number of iterations of Γ_D of interest evaluating the acceptance condition of a statement to the truth value 1 as well as completions evaluating the acceptance condition to 0 (*conti*). Secondly, for when all completions evaluate the acceptance condition to 1 (*valid*); finally for when all completions evaluate the acceptance condition to 0 (*unsat*).

$$\begin{aligned}\rho_{valid}(D) := \{&conti(S,I) \mathbin{:-} sat(S,I), inv(S,I).\\&valid(S,I) \mathbin{:-} sat(S,I), not\ inv(S,I).\\&unsat(S,I) \mathbin{:-} inv(S,I), not\ sat(S,I).\}\end{aligned}$$

Now computation of the admissible semantics for the ADF D corresponds to guessing an interpretation (via $\rho_{guess}(0)$), applying the characteristic operator Γ_D to the guessed interpretation once (this is indicated by the fact *step*(0)), and checking whether the guessed interpretation is compatible with the application of Γ_D to it (i.e. for the guessed interpretation v, $v(s) \leq_i \Gamma_D(v)(s)$ for every $s \in S$; equivalently if $v(s) = x$ then there is some $w \in [v]_2$ s.t. $w(\phi_s) = x$ for $x \in \{1, 0\}$). Finally, the predicate *asg* with two arguments rather than three is used to set the output of the program.

$$\begin{aligned}\rho_{adm}(D) :=& \pi_{arg}(D) \cup \{step(0).\} \cup \rho_{guess}(0) \cup \rho_{sat}(D) \cup \rho_{valid}(D) \cup \\ & \{\,\texttt{:-}\, asg(S,1,0), inv(S,0).\ \texttt{:-}\, asg(S,0,0), sat(S,0).\} \cup \\ & \{asg(S,1) \,\texttt{:-}\, asg(S,1,0).\ asg(S,0) \,\texttt{:-}\, asg(S,0,0).\ asg(S,\mathbf{u}) \,\texttt{:-}\, asg(S,\mathbf{u},0).\}\end{aligned}$$

The resulting encoding is very similar and, in fact, slightly more cumbersome to the encoding $\pi_{adm}(D)$ we presented in Section 3.2.1. More significant differences arise when using the `DIAMOND` style encodings for semantics such as the grounded and stable for which in sections 3.2.4 and 3.2.5 respectively we gave encodings more clearly deviating from their definition in terms of the characteristic operator.

As an example, let us consider the encoding for the grounded semantics in the `DIAMOND` style. The encoding is based on the incremental application of the characteristic operator Γ_D starting with the minimally informative interpretation (w.r.t $\leq_i$) that assigns the truth value $\mathbf{u}$ to all statements of the ADF D. So, in this case, rather than with a guess, we start with assignments to statements corresponding to the minimally informative interpretation:

$$\rho_u(D) := \{asg(s, \mathbf{u}, 0).\mid s \in S\}$$

The predicate *step* needs to be initialised and a maximum number of applications of the characteristic operator needs to be set to avoid infinite groundings of the encoding for the grounded semantics. The latter is done by using the predicate *maxit* ($|S|$ is the maximum number of iterations needed to reach a fix point of Γ_D)

$$\rho_i(D) := \{step(0).\ maxit(|S|).\}$$

The interpretation obtained by one step of the application of Γ_D is computed using the predicates defined in $\rho_{valid}(D)$:

$$\begin{aligned}\rho_{nasgn} := \{& asg(S,1,I+1) \,\texttt{:-}\, valid(S,I).\\ & asg(S,0,I+1) \,\texttt{:-}\, unsat(S,I).\end{aligned}$$

$$asg(S, \mathbf{u}, I+1) \,\texttt{:-}\, conti(S, I).\}$$

The following program fragment then computes the steps required for computing the minimal fixpoint of Γ_D as well as when a fixpoint has been reached (the latter via the predicate *fp*):

$$\begin{aligned}
\rho_{fp} := \{&fp(I) \,\texttt{:-}\, step(I), fp(S, I), arg(S).\\
\\
&fp(S, I+1) \,\texttt{:-}\, asg(S, 1, I+1), asg(S, 1, I).\\
&fp(S, I+1) \,\texttt{:-}\, asg(S, 0, I+1), asg(S, 0, I).\\
&fp(S, I+1) \,\texttt{:-}\, asg(S, \mathbf{u}, I+1), asg(S, \mathbf{u}, I).\\
\\
&step(I+1) \,\texttt{:-}\, step(I), not\ fp(I), not\ maxit(I).
\end{aligned}$$

The grounded interpretation has been computed whenever a fixpoint has been reached; alternatively, the maximum number of interpretations of Γ_D has been reached. The following program fragment thus sets the output predicates accordingly:

$$\begin{aligned}
\rho_{out} := \{&asg(S, 1) \,\texttt{:-}\, asg(S, 1, I), fp(I).\\
&asg(S, 0) \,\texttt{:-}\, asg(S, 0, I), fp(I).\\
&asg(S, \mathbf{u}) \,\texttt{:-}\, asg(S, \mathbf{u}, I), fp(I).\\
\\
&asg(S, 1) \,\texttt{:-}\, asg(S, 1, I), maxit(I).\\
&asg(S, 0) \,\texttt{:-}\, asg(S, 0, I), maxit(I).\\
&asg(S, \mathbf{u}) \,\texttt{:-}\, asg(S, \mathbf{u}, I), maxit(I).\}
\end{aligned}$$

The dynamic `DIAMOND` style encoding for the grounded semantics is thus as follows:

$$\begin{aligned}
\rho_{grd}(D) :=&\pi_{arg}(D) \ \cup\ \rho_i(D) \ \cup\ \rho_u(D) \ \cup\ \rho_{sat}(D) \ \cup\\
&\rho_{valid}(D) \ \cup\ \rho_{nasgn} \ \cup\ \rho_{fp} \ \cup\ \rho_{out}
\end{aligned}$$

While the encoding $\pi_{grd}(D)$ is arguably more declarative and simpler, the encoding $\rho_{grd}(D)$ more clearly follows the (standard) definition of the grounded semantics and avoids the long rule in $\pi_{cm}(D)$ used to compute a counter-model to the guessed interpretation being the right candidate for the grounded interpretation of the ADF D in the encoding $\pi_{grd}(D)$. In conclusion, both encoding styles have their a-priori advantages and disadvantages; empirical studies may aid in further discriminating between them in practice.

3.2.7 Implementation and experiments

We have implemented a system which, given an ADF, generates the encodings for the ADF presented in sections 3.2.1 to 3.2.5. The system, YADF ("Y" for "dynamic"), is publicly available[14] and currently (version 0.1.1) supports the admissible, complete, preferred, and stable semantics. As is the case of our system QADF, it is implemented in Scala[15] and can, therefore, be run as a Java[16] executable.

The input format for YADF is the input format that has become the standard for ADF systems and has already been presented in Section 3.1.4 (minus the information about the link types which YADF does not support). A typical call of YADF (using a UNIX command line) looks as follows:

```
java -jar yadf_0.1.1.jar -adm -cred a filename | \
      ./path/to/lpopt | ./path/to/clingo
```

Here we ask YADF for the encoding of credulous reasoning w.r.t. the admissible semantics for the ADF specified in the file specified via *filename* and the statement a. As hinted at in the introduction to this work, using the rule decomposition tool lpopt[17] [BMW16b] is recommended for larger ADF instances. We have tested YADF using the ASP solver clingo [GKK$^+$18][18]. We provide the complete usage (subject to change in future versions) of YADF:

```
usage: yadf [options] inputfile
with options:
 -h             display this help
 -adm           compute the admissible interpretations
 -com           compute the complete interpretations
 -prf           compute the preferred interpretations
 -stb           compute the stable interpretations
 -cred s        check credulous acceptance of statement s
 -scep s        check sceptical acceptance of statement s
```

We have reported on an empirical evaluation of the performance of YADF w.r.t. some of the main alternative ADF systems available: DIAMOND [ES14] (static ASP based system; version 0.9) and QADF [DWW14] (QBF based system; version 0.3.2) in [BDH$^+$17]. After this work, there have been three further experimental evaluations of ADF systems (including YADF) [Kes17, DKLW18, LMN$^+$18a] (two in which we collaborated) largely

[14] https://www.dbai.tuwien.ac.at/proj/adf/yadf/
[15] https://www.scala-lang.org/
[16] https://www.java.com
[17] https://www.dbai.tuwien.ac.at/research/project/lpopt/
[18] https://potassco.org/clingo/

based on the setup of our experiments for [BDH$^+$17], but also considering newer systems: the newest variant of DIAMOND, goDIAMOND [SE17], as well as the incremental SAT-based system k++ADF [LMN$^+$18a]. The setup and results of our study for [BDH$^+$17] are detailed in Appendix B. Moreover, in Section B.3 of Appendix B we survey all recent empirical evaluations of ADF systems and reassess the results from [BDH$^+$17] in light thereof.

The general conclusion, as also expressed at the end of Appendix B, is that while our experiments from [BDH$^+$17] suggested YADF to be the better performing of the then considered systems (DIAMOND 0.9 and QADF 0.3.2) this picture has changed with subsequent experiments. In particular, the clearly overall best performing approach for ADF systems seems to be, at current moment, the incremental SAT-based approach implemented in the system k++ADF (despite the fact that even this system still has quite a few time-outs when applied to dense ADFs for the preferred semantics). But even just considering ASP-based systems, while competitive for the admissible semantics, YADF is clearly behind in performance w.r.t. goDIAMOND on the ADFs considered in recent investigations.

Some hope is provided by results on very dense ADFs considered in the experiments, where the constraint built into goDIAMOND of not supporting ADFs with statements having more than 31 parents (given the need for translating ADFs with propositional acceptance conditions to acceptance conditions as boolean formulas) is reflected in the constant number of time-outs on all reasoning tasks (admissible and preferred) as well as on acyclic and non-acyclic instances. While YADF also has quite a few time-outs (in fact, a few more than goDIAMOND), there is some increase in performance on the acyclic instances (and better average run-times than for goDIAMOND) which suggests room for improvement.

CHAPTER 4

Towards an ASP based controlled natural language interface to argumentation

In this section we develop the design of a controlled natural language (CNL) [Kuh14] based interface to argumentation, that ultimately relies on ASP. Specifically we motivate and show the design of an implementation of what co-authors of ours have dubbed the EMIL (acronym for "Extracting Meaning from Inconsistent Language") pipeline [SWD]. The input of the pipeline in question are collections of rules in an extension of an existing CNL, ACE [FKK08], allowing for the expression of potentially conflicting defeasible rules in the form of normality assumptions in addition to strict rules. Such rules are, whenever possible, translated into defeasible theories or programs as defined in Section 2.4 and can, hence, be made sense of in terms of the direct stable semantics also defined in the latter section. Finally, stable sets are verbalised using ACE.

The design of the system we develop here is based on our experience with an initial experimental prototype implementing EMIL we developed relying heavily on an existing interface to rule systems for ACE, `ACERules` [Kuh07]. The nature of this initial prototype is sketched in Section 4.2.2. We encountered several issues with this system which led to an in-depth investigation of the inner workings of `ACERules`; the results of which forms a large part of this chapter. Specifically in Section 4.1 we give a high level description of the system `ACERules`, paying attention not only to features we have adopted in our design but particularly to the sources of the problems we found. Note that such a description is unavailable to date; the only publications on `ACERules` [Kuh07, Kuh10] deal with so called grouping (see Section 4.1.3) and in a less detailed manner than we do here. Our description is, nevertheless, based on an analysis of the very transparent and quite well documented (although the accompanying documentation is somewhat outdated) source code of `ACERules`.

The system `ACERules` basically works by filtering ACE texts corresponding to general rules allowing, for instance, existentially quantified variables as well as conjunctions of atoms in the heads and bodies of rules. In particular, these atoms can appear negated (either via strong negation or negation-as-failure). A large part of the system `ACERules` consists in several transformations attempting to make such general rules, which we call ∃-rules [GGLS15], conform to the format of the rule systems it provides an interface to (e.g. normal logic programs as defined in Section 2.2). The issues we encountered when using the system `ACERules` are with these transformations. Summarising, the problem is that in some cases the transformations introduce semantic errors in the interpretation of ACE texts. On the other hand, providing an interface to rule systems while preserving the semantics, leads to many natural ACE texts expressed as rules being filtered-out when they could be handled with modern-day rule systems (as we show in Section 4.2.4).

In Section 4.2 we outline the already alluded to alternative design for an implementation of EMIL. The crucial aspect of this implementation is that we target ∃-rules, also including defeasible in addition to strict ∃-rules, rather than normal rules. Our translation of such ∃-rules to normal rules, which we further discuss as well as sketch in Section 4.2.4.1, allows us to circumvent the main problems we found with `ACERules` in a uniform manner; on the other hand we adopt features of `ACERules` we found to be of benefit. We show encodings for evaluating defeasible theories via the direct stable semantics using ASP in Section 4.2.4.2. In contrast to previous existing encodings [SW17], these encodings are optimised for defeasible theories with variables as well as function symbols (which we make use of in our translation of ∃-rules to normal rules). In Section 4.3 we give an extended example of the use of an implementation of the EMIL pipeline in motivating this pipeline in the context of ACEWiki [Kuh09], a prototype of a Wikipedia style encyclopedia expressed in ACE.

Before turning to the description of the features and limitations of the system `ACERules` and the design of our novel implementation of the EMIL pipeline we give a very brief overview of the CNL ACE highlighting aspects important for our work (Section 4.0.1). In Section 4.4 we present the functioning of an initial prototype, `emil`, we implemented based on the ideas developed in this chapter as well as the results of some preliminary experiments. Given the (practical) complexity of the tasks studied in this chapter and the more exploratory nature of the work, we turn here to a more informal presentation than in the previous chapters.

4.0.1 ACE: a brief overview

Attempto Controlled English or ACE for short, is a controlled natural language for English with an unambiguous translation to first order logic[FKK08]. We here refer to ACE version 6.7, the last version of ACE available at the time of writing this work. The vocabulary of ACE is composed of a fixed set of function words (determiners, quantifiers, coordinators, negation words, pronouns, query words, modal auxiliaries, copula be, and Saxon genitive marker 's) as well as a fixed set of predefined phrases (e.g. "there is", "it is false that”). ACE content words, which are user defined and extendable, are nouns, adjectives, adverbs, and prepositions.

Main building blocks for ACE texts are simple sentences having the structure: subject + verb + complements + adjuncts. Complements (direct and indirect objects) are necessary for transitive verbs (e.g. “insert something”) and ditransitive verbs (give something to somebody, give somebody something). Adjuncts (adverbs, prepositional phrases), on the other hand, are optional. Verb phrases are written in third person singular or plural, present tense, and active or passive voice. Composite ACE sentences are recursively built from the simple sentences using coordination (e.g. and/or), subordination (e.g. if/then sentences, negation) , quantification (e.g. “there are”, “for each”), and negation. ACE also includes constructs for expressing queries and commands. The construction rules for ACE are detailed in [ace13a].

ACE texts (sequences of ACE sentences) are interpreted using a deterministic set of interpretation rules; these are detailed in [ace13b]. E.g. the ACE sentence (example from [ace13b])

```
A customer inserts a card that is valid and opens an account.
```

which in English is ambiguous between the customer or the card opening the account, in ACE has the fixed meaning that the customer opens the account. In order to express that the card opens the account the following alternative needs to be used (note the “that” before “opens an account”):

```
A customer inserts a card that is valid and //
                                       that opens an account.
```

As a further important example, also constructs involving anaphoric references are interpreted in a uniform manner by the interpretation rules of ACE; more precisely, all anaphoric references are identified with the most specific accessible noun phrase that agrees in gender and number. E.g. given the sentence (again, the example is from [ace13b])

```
A customer enters a red card and a blue card.
```

then

```
The card is correct.
```

refers to the blue card while

```
The red card is correct.
```

refers to the red card.

Formally, the semantics of ACE texts is defined in terms of discourse representation structures (DRSs)[BB05]. DRSs are constructed dynamically to support anaphora resolution (e.g. associating pronouns with their referents), but once computed (i.e. all anaphoric references are resolved) can be translated to first oder logical formulas. To simplify matters we will, hence, in this work treat DRSs and their first order (FOL) representations more or less synonymously, and only show the first order representation of DRSs corresponding to ACE texts we consider.

The DRSs (or first order formulas) used to interpret ACE texts [FKK13] make use of a reified or "flat notation" for logical atoms. E.g. a sentence such as "Mary gives John a present" which in first order logic textbooks would customarily be represented more or less as

$$\exists A(present(A) \wedge give(mary, A, john))$$

is, rather represented as:

$$\begin{aligned}\exists A, B\big(&object(A, present, countable, na, eq, 1)\\ &\wedge\\ &predicate(B, give, named(Mary), A, named(John))\big)\end{aligned}$$

where nouns and verbs are "wrapped" into the special atoms "object" and "predicate"; in particular with "object" including further semantic information ("countable","na","eq","1") pertaining to the semantics of the noun phrase ("a present") in question (e.g. "countable" indicates that "present" is a countable rather than a mass noun)[1]. Crucially, verb phrases are given a Neo-Davidsonian event-theoretic semantics [Par90], thus introducing a referent (quantified variable; in the example "B") allowing to attach modifiers stemming from adverbs and prepositional phrases used in ACE texts as for the simple ACE sentence "Mary reluctantly gives John a present" which has the interpretation

$$\exists A, B\big(object(A, present, countable, na, eq, 1)$$

[1]The parse of ACE texts given by the parser for ACE `APE` also includes additional elements to refer to the position in the ACE text in which the different constituents of phrases are introduced. For simplicity, we ignore these in what follows.

$$\begin{array}{l}\wedge\\ modifier\_adv(B, reluctantly, pos)\\ \wedge\\ predicate(B, give, named(Mary), A, named(John)))\end{array}$$

Note, in particular, the logical atom “modifier_adv” used to attach the (non comparative, i.e. positive) adverb “reluctantly” to the event of “giving” introduced by the sentence. Specialised atoms are used for noun phrases (“object”), adjectives (“property”), relations introduced by the of construct (“relation”), relations introduced by different forms of verbs (“predicate”), modifiers introduced by adverbs (“modifier_adv”), modifiers introduced by prepositional phrases (“modifier_pp”), for the distributive reading of plural noun phrases (“has_part”), and queries (“query”). Critical for our purposes is that only noun phrases and verb phrases introduce referents (quantified variables) in DRSs; the remaining predicates make use of the referents introduced by noun and verb phrases. Note also that named entities e.g. “Mary” are wrapped inside of the “named” function, e.g. “named(Mary)”. For details on the form of the DRSs that underly the semantics of ACE we refer to [FKK13].

Beyond the appropriateness of ACE for basic knowledge representation, a large part of the attraction of ACE comes from the many open source tools that are available, also in the form of web-services. The main workhorse for ACE is the parser `APE`, which among many other things parses ACE texts into DRSs and also offers translations of the DRSs into several other formats including FOL. It also offers paraphrasing of ACE texts (which eases composing ACE texts with the intended meaning) as well as verbalisation of DRSs into ACE, specifically Core ACE and NP ACE [Kal09]. `APE` comes bundled together with a lexicon of around 100,000 entries for content words; it can also guess the category of words not in the lexicon but as an alternative words can be preceded with a label indicating the category. E.g. “a:extramundane n:eviternity” indicates that “extramundane” is an adjective and “eviternity” a noun.

The main reasoning system available for ACE is `RACE` [Fuc10] which adapts the theorem prover `Satchmo` [MB88] to check consistency, textual entailment and answer queries for ACE texts. `RACE` supports all of `ACE` but is, therefore, also rather inefficient in practice. The basis of our work, as already indicated, is the system `ACERules` [Kuh07] which offers an ACE interface to rule systems. We will describe this system in detail in Section 4.1. Most tools for ACE, including `APE`, `RACE`, and `ACERules` are implemented in `Prolog` [Bra14]. The versions of `APE` and `ACERules` we refer to in this work are versions 6.7-131003 and 2008-11-24 respectively. We also made use of the `RACE` web-client[2]. We refer to the webpage of the Attempto project for further resources, tools, as well as more detail on the tools available for ACE[3].

[2]At `http://attempto.ifi.uzh.ch/race/`. Last accessed on 2019-01-08.

[3]`http://attempto.ifi.uzh.ch/site/resources/`

4.1 Design, features, and limitations of the system ACERules

AceRules builds on APE to provide an ACE-based interface to formal rule systems. More specifically, the target formal rule systems are logic programs under the stable [GL88] (section 2.2) as well as the courteous semantics [Gro97]. Thus, the rule systems are as the logical programs described in Section 2.2; yet without function-symbols and having strong negation, written with the symbol $\neg$, in addition to negation as failure. Strong negation allows to express that some fact (written as an atom) is provably not the case [GL91]. Evaluation via the courteous semantics also allows defining priorities over rules but is, on the other hand, only defined for a subset of logical programs: acyclic programs. AceRules relies on external solvers for evaluation via the stable semantics and includes a native implementation of the courteous semantics. As already indicated, AceRules is implemented in Prolog.

A large part of AceRules consists in taking the DRS output as given by APE and applying a series of transformations with the goal of making this output conform to the syntax of logical rules. We now describe in some detail each of these transformations more or less in the order in which they take place in AceRules, also pointing out some bugs and limitations we found. This section presents an attempt at "reverse-engineering" a high level yet useful -, for analytical purposes,- description from the very well structured and commented source code of AceRules. As we already indicated, grouping, which we also describe in this section (Section 4.1.3), is considered (in less detail) in [Kuh07].

4.1.1 Filtering relevant DRSs

Whenever these appear in the input ACE text, ACERules starts by processing meta-structures (labels and overrides statements) which can be used by the courteous semantics. We don't consider this step here as it is tangential to our work (although they play a role in the prototype we implemented that is described in Section 4.2.2). Then the system applies APE to obtain a DRS corresponding to the input text.

As a first step relevant for our work, AceRules checks that the DRS that results from applying APE to the input ACE text has a syntactic structure amenable to further transformations. The syntactic structure in question is that on the one hand the DRS contains conditions which are "wide literals". These correspond to either a conjunction of atoms (including a single atom) or strict negation applied to a conjunction of atoms. Wide literals can also contain "flat modality boxes" in the place of atoms. The latter, whose further treatment we ignore in what remains for being extraneous to our work, are modality operators applied to conjunctions of atoms.

On the other hand, the DRSs that result from applying APE to the input ACE text can also contain conditions corresponding to implications $B \rightarrow H$ having exactly one wide literal in the head H as well as several (possibly, just one) wide literals in the body B. The wide literals in the body B can also appear nested within a negation as failure DRS.

We call the latter "naf wide literals" (these hence include simple wide literals; i.e. those which don't appear nested within a negation as failure DRS). Implications of the form $((B_1 \to B_2) \to H)$ with H corresponding to a single wide literal and B_1 as well as B_2 to conjunctions of naf wide literals are transformed to conditions representing the equivalent implications $(B_1 \wedge B_2) \to H$.

In conclusion, ACE texts that survive the first check of the DRSs obtained from `APE` by `ACERules` are those which (after transformation of rules with double implication) are parsed as collections of rules having wide literals in the head and naf wide literals in the body. The rules in question can also have empty bodies; these are simply wide literals. We call such rules ∃-rules, since they have more or less the form of the rules of ∃-ASP (excluding strong negation) as defined in [GGLS15].

4.1.2 Condensation and other transformations of the atoms of the DRSs

4.1.2.1 Transformations of atoms introduced by nouns

Right after having filtered DRSs relevant for further transformation, `ACERules` applies a series of transformations at the level of the predicates of the DRSs. First of all, `APE` produces special atoms standing for indefinite pronouns. These are the predicate

$$object(A, somebody, countable, na, eq, 1)$$

for "someone/somebody" (negated for "no one/ nobody") and the predicate

$$object(A, something, dom, na, na, na)$$

for "something" (negated for "nothing"). `ACERules` removes such predicates[4], replacing them with predicates standing for an "anonymous object"

$$object(A, V_1, V_2, V_3, V_4, V_5)$$

and where each V_i $(1 \leq i \leq 5)$ is a fresh variable not appearing elsewhere in the DRS. Also, the V_is are quantified in accordance to where they appear in the DRS.

[4]Strictly speaking `ACERules` replaces any predicates of the form $object(A, somebody, Q, na, eq, 1)$ and $object(A, something, Q, na, na, na)$ where Q can be any constant. We assume this is to also cover the "named" class of objects which is introduced by `ACERules` to represent objects introduced by proper names as we explain furtheron.

Example 11. *The sentence "Everybody loves something." which by* `APE` *gets parsed as*

$$\forall A(object(A, somebody, countable, na, eq, 1) \rightarrow \\ \exists BC(object(B, something, dom, na, na, na) \land predicate(C, love, A, B)))$$

is transformed (ignoring later transformations such as predicate condensation and skolemisation which remove all existentially quantified variables) by `ACERules` *into*

$$\forall ABCDEF(object(A, B, C, D, E, F) \rightarrow \\ \exists GHIJKLM(object(G, H, I, J, K, L) \land predicate(M, love, A, G)))$$

This transformation has, on the one hand, the positive effect of making every statement involving an indefinite for persons applicable to all persons; e.g. from the text

```
Mary is a person.
Everyone loves something.
```

it follows without further ado that "Mary loves something". Also, every statement involving an indefinite for things is applicable to things; e.g. from the text

```
There is a table.
Everything is physical.
```

it follows that "The table is physical". The problem is clearly that the transformations of `ACERules` blur the distinction between inanimate objects and persons. E.g. from the text

```
There is a table.
Everybody likes Mary.
```

the current version of `ACERules` returns the result[5] (under e.g. the courteous semantics):

```
There is a table X1.
The table X1 likes Mary.
Mary likes Mary.
```

[5]In defense of `ACERules` it should be mentioned that `RACE` suffers from a similar problem; thus posing the query "Is there a table that likes Mary?" results in `RACE` answering yes.

Related to the transformation of the predicates standing for "anonymous" objects, special (skolem-) constants are introduced by APE when proper names are mentioned in a text. E.g. "John waits" is represented as

$$\exists A(predicate(A, wait, named(John)))$$

where the constant $named(John)$ stands for "John". ACERules adds an object predicate for any such constant standing for a proper name it finds. E.g. atoms standing for the conjunction

$$object(named(John), John, named, na, eq, 1) \wedge$$
$$object(named(Mary), Mary, named, na, eq, 1)$$

are added to the DRS for a text containing "John waits for Mary at the train-station". This allows e.g. the rule "everybody likes something", in the transformed rendering given by ACERules, to also be applicable to Mary and John. Note that here the class "named" has been added to the possible classes for objects $\{\text{dom}, mass, countable\}$ introduced by APE [FKK13].

In conclusion, replacing predicates standing for indefinite pronouns with a predicate standing for anonymous objects introduces errors. Although this is a "quick fix" for dealing with many ACE texts, a cleaner solution seems to us to be to not modify the predicates given by APE and rather introduce additional rules for specifying the desired relations. These can be added either at system or user-level. E.g. the parse given by APE for the sentences

```
Mary is a person.
Every person is somebody.
Everyone loves something.
```

introduces the right relations to allow the inference that "Mary loves something". Introduction of appropriate rules at system level would require further semantic information about noun phrases than what is currently available as part of the output of APE[6].

4.1.2.2 Predicate condensation

As we have explained in Section 4.0.1, APE produces reified or flat notations for logical atoms. In particular, verbs and their modifiers (adverbs and prepositional phrases) are

[6]Note that, in particular, not all named objects are "somebodies" as, for instance, cities and dogs can have names as well.

given an event-theoretic semantics. I.e. verbs introduce referents standing for events and modifiers are then predicated on such events in a manner very much resembling the way adjectives are properties of objects introduced by noun phrases.

The problem for `ACERules` is, first of all, that this manner of handling verb phrases introduces several atoms linked together by conjunction (i.e. wide literals rather than literals as required by logical rules). A more serious issue is that the event-theoretic semantics introduces referents or quantified variables (over events). Often these are implicitly existentially quantified or are introduced within negations. This is in contrast to the input languages of the rule systems `ACERules` links to which only allow a restricted form of quantification: universal quantification over variables introduced by positive atoms in the bodies of rules.

Predicate condensation is the solution implemented by the system `ACERules` to deal with the aforementioned issues. Roughly, the solution is to aggregate atoms referring to verbs with those of their modifiers while removing the variables introduced to refer to the events denoted by verbs.

Example 12. *The sentence "Brutus unhesitatingly stabs Caesar in the back with a knife." (adapted from [Par90]) gets parsed by* `APE` *as*

$$\begin{aligned}\exists ABC(&object(C, back, countable, na, eq, 1) \wedge object(A, knife, countable, na, eq, 1)\\ &\wedge modifier\_adv(B, unhesitatingly, pos)\\ &\wedge predicate(B, stab, named(Brutus), named(Caesar))\\ &\wedge modifier\_pp(B, with, A) \wedge modifier\_pp(B, in, C))\end{aligned}$$

`ACERules` *removes the variable B introduced for the verb "stab" and groups together the predicates for the verb and its modifiers in a new predicate pred_mod, where the last argument includes a list with the modifiers:*

$$\begin{aligned}\exists AC(&object(C, back, countable, na, eq, 1) \wedge object(A, knife, countable, na, eq, 1)\wedge\\ &pred\_mod(stab, named(Brutus), named(Caesar),\\ &\qquad [modifier\_pp(in, C), modifier\_pp(with, A),\\ &\qquad modifier\_adv(unhesitatingly, pos)]))\end{aligned}$$

The example is for a transitive verb ("stab"). Intransitive and ditransitive verbs are treated in analogous fashion.

Clearly, predicate condensation also transforms the semantics associated with verb phrases and their modifiers by `APE`. In particular, the "diamond inference pattern" that holds between a verb phrase plus modifiers and its components (see [Par90]) is broken. For instance in the following group of sentences the first sentence implies all of the sentences below it, the second sentence implies the last sentence, and each sentence from the third onwards implies each sentence below it.

```
Brutus unhesitatingly stabs Caesar in the back with a knife.
Brutus unhesitatingly stabs Caesar.
Brutus stabs Caesar in the back with a knife.
Brutus stabs Caesar in the back.
Brutus stabs Caesar with a knife.
Brutus stabs Caesar.
```

This inferential pattern is captured by the reading of the first sentence given by APE but not by ACERules (see Example 12). So, for example, the output given by ACERules for the following text

```
Brutus unhesitatingly stabs Caesar in the back with a knife.
If Brutus stabs Caesar then Brutus is a traitor.
```

does not include the assertion that Brutus is a traitor[7].

Another issue with predicate condensation which is more likely a bug than an intended feature of ACERules is that modifiers in the *pred_mod* predicates are aggregated into ordered lists, while their semantics (at least as given by the parse by APE) would require them to be aggregated into sets. So, for instance, the output given by ACERules for the following text

```
Brutus unhesitatingly and violently stabs Caesar.
If Brutus unhesitatingly and violently stabs Caesar
                              then he is a remorseless traitor.
```

is

```
Caesar is a:remorseless .
Caesar is a n:traitor .
Brutus v:stabs Caesar unhesitatingly and violently.
```

while the result for the equivalent text (note the re-ordering of the adverbs in the second sentence)

[7]While, for instance, RACE finds a proof of “Brutus is a traitor” from the same text.

```
Brutus unhesitatingly and violently stabs Caesar.
If Brutus violently and unhesitatingly stabs Caesar
                              then he is a remorseless traitor.
```

is

```
Brutus v:stabs Caesar unhesitatingly and violently.
```

I.e. the rule contained in the input text has not been applied.

4.1.2.3 Postprocessing of condensed predicates for the copula "be"

The copula "be", being one of the most frequently used words in the English language[8], has several distinct uses. On the other hand `APE` leaves it underspecified; treating the copula as just another verb.

According to [Fuc10] in ACE the copula "be" can be used in at least one of three ways. First of all, to express identity as in "John is Harry". Secondly, to express class membership as in "Every cat is an animal". Third, for predication as in "The cat is black" or "John is in the garden." `ACERules` transforms condensed predicates (as explained in Section 4.1.2.2) involving the copula "be" in order to support reasoning in accordance with some of the distinct uses of the copula.

In the first place, there is a transformation if the verb "be" is used in its transitive form, without any modifiers (i.e. adverbs and prepositional phrases), and links a variable or constant to a variable. In this case the copula is interpreted by `ACERules` as identity. After predicate condensation in the detailed case the system produces a predicate of the form

$$pred\_mod(be, A, B, [])$$

(note the empty list in the fourth argument) where B is a variable and A is either a variable or constant. `ACERules` replaces all occurrences of the variable "B" with "A" (via Prolog unification) and removes the condensed version of the predicate representing the copula. According to the (outdated) documentation accompanying `ACERules` this is to treat sentences like "John is a customer"; i.e. instances where "be" is used to express class membership. In reality the implementation extends beyond this use of the copula "be", covering any use where the copula is not modified by adverbs or prepositional phrases except for the case where the copula links two proper names.

[8]See e.g. `https://www.wordfrequency.info/free.asp?s=y` (accessed 21.11.2018))

Example 13. *The sentence "John is a customer." gets parsed by* `APE` *as*

$$\exists AB\big(object(A, customer, countable, na, eq, 1) \wedge predicate(B, be, named(John), A)\big)$$

`ACERules` *transforms the parse into*

$$object(named(John), customer, countable, na, eq, 1).$$

An example of class membership expressed as a rule is "Every cat is an animal" which gets parsed by `APE` *as*

$$\begin{aligned}&\forall A\big(object(A, cat, countable, na, eq, 1) \rightarrow\\&\quad \exists BC\big(object(B, animal, countable, na, eq, 1) \wedge predicate(C, be, A, B)\big)\big).\end{aligned}$$

This is transformed by `ACERules` *into*

$$\forall A\big(object(A, cat, countable, na, eq, 1) \rightarrow object(A, animal, countable, na, eq, 1)\big).$$

Finally, for an example involving predication via adjectives consider the sentence "The cat is black". This sentence gets parsed by `APE` *as*

$$\exists ABC\big(object(C, cat, countable, na, eq, 1) \wedge property(A, black, pos) \wedge predicate(B, be, C, A)\big)$$

and gets transformed by `ACERules` *into*

$$\exists A\big(object(A, cat, countable, na, eq, 1) \wedge property(A, black, pos)\big).$$

`ACERules` disallows the use of the copula "be" to link two proper names as in "John is Harry"[9]. An error message is displaced in such a case.

[9] This is in contrast to `RACE` which, although assumes that different names denote different individuals (unique name assumption) by default, does allow the use of the copula "be" to state exceptions to this default assumption. Note also that the vacuous sentence "John is John" easily could be allowed but is not supported by `ACERules`.

A further -, to us somewhat mysterious,- transformation that `ACERules` applies to condensed predicates involving the copula "be" is for when "be" has modifiers (prepositional phrases and adverbs) and also involves adjectives. This is the case, for instance, when after predicate condensation there are predicates of the form

$$\begin{aligned} &pred\_mod(be, A, B, Mods)\\ &\quad property(B, Adj, D) \end{aligned}$$

in the conditions of the DRSs. Here $Mods$ is a list of modifiers and Adj is a name of an adjective. Such predicates are merged into a new predicate as follows:

$$be\_adj(A, Adj, D, Mods).$$

Example 14. *As already hinted at, the described transformation carried out by `ACERules` is somewhat of a mystery to us since it doesn't seem to cover nor capture any meaningful class of statements. An example would be, for instance, the "The cat a:clearly is black." (note the need for indicating to `APE` the class of the word "clearly"). For this sentence `APE` produces the parse:*

$$\begin{aligned} \exists ABC\big(&object(C, cat, countable, na, eq, 1)\wedge\\ &modifier\_adv(B, clearly, pos) \wedge property(A, black, pos)\wedge\\ &predicate(B, be, C, A)\big). \end{aligned}$$

This parse gets transformed by `ACERules` to

$$\begin{aligned} \exists A\big(&be\_adj(A, black, pos, [modifier\_adv(clearly, pos)])\wedge\\ &\quad object(A, cat, countable, na, eq, 1)\big) \end{aligned}$$

Our suspicion is that the original intention was for the predicate *be_adj*, similarly to what happens in `RACE` [Fuc10], to capture the use of "be" to link a proper name or a noun phrase to an adjective[10]. Yet this is not the case in the current implementation of the system (see the reading given by `ACERules` for the sentence involving "the cat is black" in Example 13).

[10]Possibly this was modified later or a possible bug was not found given that the implementation works well for the case when "be" links a proper name or a noun phrase to an adjective. These adjective will, as far as we can tell, always be intersective; see discussion furtheron.

It is indispensable, for purposes of automating reasoning from ACE texts, to distinguish between the different uses of the copula "be"; in particular, those cases when the copula is modified by adverbs or prepositional phrases, those where the copula links two noun phrases, as well as the case where the copula is used for simple predication (linking a noun phrase to an adjective). As far as we can tell, ACERules indirectly distinguishes cases (such as "John is in the garden") where the copula is solely modified by adverbs or prepositional phrases indirectly by leaving these cases underspecified (i.e. giving "be" the same treatment as other verbs). ACERules disallows the use of the copula "be" to link two proper names. Finally, in practice ACERules (as indicated previously, this may be unintended) treats most usual meaningful uses of the copula "be" to express class membership and predication by the transformation illustrated in Example 13.

We note that the latter transformation implicitly amounts to an "intersective" reading of adjectives and introduces errors in cases where adjectives are not intersective. As far as we can tell use of the copula "be" for simple predication of a property of an object can be argued to usually give an intersective reading. E.g. "Mary is beautiful" means Mary is one of the "beautiful things" while "Mary is kind" means that Mary is one of the "kind things" and hence, for instance, from

```
Mary is beautiful.
Mary is kind.
If Mary is beautiful and kind then she is popular.
```

it follows, as indicated by ACERules, that "Mary is popular". ACE also implicitly imposes an intersective reading of adjectives used in noun phrases (hence, without the copula "be"), in particular allowing several adjectives to be conjoined only by "and"; e.g. cumulative adjectives such as in "an unmarked police car patrols the area"[11], which are often not intersective, are disallowed.

Problems arise when adjectives are used in the context of expressing class membership. Here APE correctly leaves the "logic" of adjectives underspecified (to a larger degree) while the transformations carried out by ACERules force an intersective reading. Hence the output of ACERules for the following text involving the intersective adjective "yellow"[12]

```
The n:company-car is a yellow bus.
The n:company-car is a n:Volkswagen.
If the n:company-car is a yellow n:Volkswagen then the employees
                                                    are happy.
```

[11]Example adapted from https://study.com/academy/lesson/cumulative-adjectives-definition-order-examples.html. Accessed on 27.11.2018.

[12]All our examples involving different classes of adjectives are (adapted) from https://www3.nd.edu/~jspeaks/courses/2012-13/43916/handouts/13-modifiers.pdf. Accessed on 28.11.2018

correctly includes (a paraphrase of) the sentence that "the employees are happy". On the other hand RACE, which follows the underspecified reading produced by APE, judges the following related text to be consistent[13]:

```
The n:company-car is a yellow bus.
The n:company-car is a n:Volkswagen.
If the n:company-car is a yellow n:Volkswagen then the employees
                                                        are happy.
The employees are not happy.
```

Examples showing for where the intersective reading of ACERules produces wrong results are e.g. the text

```
Ralph is a former basketball-player.
Ralph is a teacher.
If Ralph is a former teacher then he does not teach.
```

which involves the non-predicative adjective "former" and for which ACERules concludes that "Ralph does not teach". Also for the following text

```
Bob is a tall midget.
Bob is a basketball-player.
If Bob is a tall basketball-player then he plays for the NBA.
```

involving the subsective adjective "tall" ACERules generates incorrect inferences not licensed by the parse given by APE (and adopted by RACE), concluding that Bob plays for the NBA.

4.1.2.4 Condensation of atoms for the preposition "of"

As already hinted at in Section 4.0.1, a special atom standing for relations introduced by of-constructs is introduced by APE. ACERules merges such predicates with those referring to the "left hand side" object in the "of" relation.

Example 15. *The sentence "John's brother loves Mary." gets parsed by APE as*

$$\exists AB(relation(B, of, named(John)) \wedge object(B, brother, countable, na, eq, 1) \\ \wedge\, predicate(A, love, B, named(Mary))).$$

[13] Illustrating the need for additional rules expressing the properties of intersective adjectives to produce correct inferences.

ACERules transforms (including the transformation of the predicate standing for the verb "love") the parse into

$$\exists A(of\_relation(object(A, brother, countable, na, eq, 1), named(John)) \\ \wedge pred\_mod(love, A, named(Mary), [])).$$

We conjecture the main reason for this transformation applied by ACERules is to further reduce the number of predicates in the conditions of the DRSs, thus making the DRSs more amenable to representation in the form of logical rules. E.g. the sentence "If John is the brother of Paul then he is not the brother of Mary" can, after applying the transformation for the copula "be" (Section 4.1.2.3) as well as the condensation of predicates involving "of", be represented as the logical rule:

$$of\_relation(object(named(John), brother, countable, na, eq, 1), named(Paul)) \rightarrow \\ of\_relation(object(named(Paul), brother, countable, na, eq, 1), named(Mary))$$

while this is not the case for the parse of the sentence given by APE.

4.1.3 Grouping

We now consider the first transformation carried out by ACERules that is not based on semantical considerations pertaining to natural language, but rather on the logical form of APE parses only. It is also the transformation described in [Kuh07], its main purpose being to remove conjunctions of several atoms appearing in the heads of rules as well as occurring negated in the bodies of rules; specifically, whenever there are such conjunctions of atoms remaining after the transformations described in Section 4.1.2. Secondly, and more fundamentally[14], grouping removes existentially quantified variables in the bodies and heads of rules (i.e. not in facts) whenever possible.

Grouping, as predicate condensation, hence amounts to aggregating predicates and removing variables, but in contrast to predicate condensation, it is applied on groups of atoms conjoined by conjunctions appearing in the heads of rules or negated (via strict and/or negation-as-failure) in the bodies of rules. Also, certain restrictions apply. The restrictions are first of all that the removed variables do not occur outside of the group (these are either universally quantified or are existentially quantified and subject to skolemisation later on; see Section 4.1.4). Secondly, there must not be any other group of predicates in the program that match (or, in Prolog jargon, unify) the aggregated

[14]Only the occurrence of existentially quantified variables within atoms presents a serious problem for rule systems since cases where this is not true can be handled by some rewriting of the relevant rules or negative wide literals using several rules (as we show in Section 4.2.4.1). The use of grouping to also deal with these latter cases therefore unnecessarily restricts the texts that AceRules can handle.

predicates, but where the variable corresponding to the removed variable in the latter set of predicates is referred to by some other predicate that is not part of the matching group. This second restriction is because in order for grouping to succeed to aggregate predicates, e.g. in the head of an implication, it must thereafter also be possible to apply grouping on all groups of predicates occurring in the program that match with the initially aggregated predicates.

Example 16. *We consider the example from [Kuh07]:*

```
John owns a car.
Bill does not own a car.
If someone does not own a car then he/she owns a house.
```

`APE` parses this text into the first order formula

$$\begin{array}{l}
\exists A, B\big(object(A, car, countable, na, eq, 1) \wedge predicate(B, own, named(John), A)\big)\\
\quad\wedge\\
\neg\exists C, D\big(object(C, car, countable, na, eq, 1) \wedge predicate(D, own, named(Bill), C)\big)\\
\quad\wedge\\
\forall E\Big(\Big(object(E, somebody, countable, na, eq, 1) \quad \wedge\\
\quad \neg\exists F, G\big(object(F, car, countable, na, eq, 1) \wedge predicate(G, own, E, F)\big)\Big)\\
\quad \rightarrow\\
\quad \exists H, I\big(object(H, house, countable, na, eq, 1) \wedge predicate(I, own, E, H)\big)\Big)
\end{array}$$

After the `ACERules` transformations described in sections 4.1.2.1 and 4.1.2.2 the parse has the form:

$$\begin{array}{l}
object(named(Bill), Bill, named, na, eq, 1)\\
\quad\wedge\\
object(named(John), John, named, na, eq, 1)\\
\quad\wedge\\
\exists A\big(pred\_mod(own, named(John), A, []) \wedge object(A, car, countable, na, eq, 1)\big)\\
\quad\wedge\\
\neg\exists B\big(pred\_mod(own, named(Bill), B, []) \wedge object(B, car, countable, na, eq, 1)\big)\\
\quad\wedge
\end{array}$$

$$
\begin{aligned}
&\forall CDEFGH\Big(\Big(object(C,D,E,F,G,H) \;\; \wedge \\
&\quad \neg\exists I\big(pred\_mod(own,C,I,[]) \wedge object(I,car,countable,na,eq,1)\big)\Big) \\
&\quad \rightarrow \\
&\quad \exists J\big(pred\_mod(own,C,J,[]) \wedge object(J,house,countable,na,eq,1)\big)\Big)
\end{aligned}
$$

Grouping constructs single "grouping" atoms for groups of predicates representing "owns a car" appearing negated in the fourth conjunct as well as the body of the rule. Also, single atoms are created for predicates representing "owns a house" in the head of the rule (last conjunct). Then the groups of predicates subject to grouping are matched with predicates appearing elsewhere in the program (third conjunct, representing "John owns a car"), replacing these with the newly constructed grouping atoms. The result is the following rewriting:

$$
\begin{aligned}
&object(named(Bill),Bill,named,na,eq,1) \\
&\quad \wedge \\
&object(named(John),John,named,na,eq,1) \\
&\quad \wedge \\
&group([pred\_mod(own,named(John),gv(0),[]),object(gv(0),car,countable,na,eq,1)]) \\
&\quad \wedge \\
&\neg group([pred\_mod(own,named(Bill),gv(0),[]),object(gv(0),car,countable,na,eq,1)]) \\
&\quad \wedge \\
&\forall ABCDEF\Big(\Big(object(A,B,C,D,E,F) \;\; \wedge \\
&\quad \neg group([pred\_mod(own,A,gv(0),[]),object(gv(0),car,countable,na,eq,1)])\Big) \\
&\quad \rightarrow \\
&\quad group([pred\_mod(own,A,gv(1),[]),object(gv(1),house,countable,na,eq,1)])\Big)
\end{aligned}
$$

Note, in particular, the introduction of constants ("grouped variables") $gv(0)$ and $gv(1)$ standing for the previous variables A and B on the one hand I and J on the other.

An example where grouping, because of the restrictions built into ACERules, fails is on the following text (slight modification of an example in [Kuh07][15]):

[15]The original form of the example is: "Bill does not own a car. John owns a car X. Mary sees the car X." Although according to [Kuh07] grouping should fail on this example it does not in the current version of ACERules. The reason is that, as we explain in more detail later on, the matching phase of grouping in ACERules does not consider sub-groups; hence it is unable to relate the groups of predicates for "Bill owns a car" with those of "John owns a car".

```
John owns a car.
Mary sees the car.
If someone does not own a car then he/she owns a house.
```

The problem here is the impossibility to group the predicates for "owning a car" appearing negated in the body of the rule, because of the reference to a car independent of someone owning it in the second sentence in the text. `ACERules` returns an error message ("the program violates the atom-restriction") in such cases.

We found a few shortcomings in the way grouping is currently implemented in `ACERules`. In the first place, as is also documented in the `ACERules` source code and as occurs with predicate condensation (Section 4.1.2.2), groups of predicates are implemented as lists while they should be implemented as sets. E.g. for the following text

```
Bill does not own a car and a house.
If someone does not own a car and a house \\
                       then he/she owns a house and a car.
```

`ACERules` concludes that[16]

```
Bill owns a house and a car.
It is false that Bill owns a car and a house.
```

A second issue we found with the current implementation of grouping in `ACERules` is that the matching phase only considers groups of atoms that match exactly, while also sub-groups need to be considered, at least in many cases. As an example consider the following text:

```
John owns a car.
Every car is an automobile.
John does not own an automobile.
```

Here `ACERules` groups together predicates for "John owns an automobile" and then is unable to relate "owning a car" (which also includes the verb "owns") with "owning an automobile" and hence concludes that[17]

[16]Note that the construction rules of `ACE` determine that the scope of "owns" in both these sentences is over a plural entity composed of a house and a car[ace13a].

[17]Reverting the order of the sentences in the text as in: "John does not own an automobile. John owns a car. Every car is an automobile." forces `ACERules` to match the group of predicates (of cardinality 1) for "an automobile" (stemming from the rule "Every car is an automobile") first. In that case "an automobile" from the first sentence is matched and then the predicates for "John does not own an automobile" cannot be grouped. Thus, in this case, grouping fails.

```
There is an automobile X1.
John owns the automobile X1.
The automobile X1 is a car.
It is false that John owns an automobile.
```

As an example where not considering sub-groups during matching leads to incomplete inferences consider the following text:

```
Bill does not own a car.
If someone does not own a car then he/she owns a house.
Every house is a property.
If Bill owns a property then he is a proprietor.
```

Here `ACERules` is able to conclude that Bill owns a house, yet not that he owns a property and hence is a proprietor.

A third shortcoming we found is that, in the current implementation of `ACERules` the matching phase seems not to consider groups of atoms matching in the predicates contained in them but differing in the degree of generality. As an example where this is necessary consider the following text:

```
Bill does not own a vehicle.
If Bill does not own a vehicle then he does not own a car.
If someone does not own a car then he/she owns a motorcycle.
If someone owns a motorcycle then he/she owns a vehicle.
```

Here `ACERules` is unable to relate the group of predicates for “Bill does not own a vehicle” and the more general “he/she owns a vehicle” and thus concludes that:

```
Bill owns a motorcycle.
Bill owns a vehicle.
It is false that Bill owns a vehicle.
It is false that Bill owns a car.
```

In conclusion, even with the current rather liberal implementation of grouping the programs `ACERules` can transform to rule format are clearly limited. Were further restrictions needed to be applied on grouping (as our study suggests may be necessary), even less ACE texts would be amenable to treatment by `ACERules`. In Section 4.2.4.1 we will sketch an alternative form of defining grouping that avoids the problems sketched above. Moreover, some form of existential quantification seems unavoidable except in very simple rules. Therefore support of some form of existential quantification as is the case in the approach described in Section 4.2.4.1 (ideally following `ACERules` in having means of removing existentially quantified variables whenever possible) also is desirable.

4.1.4 Top level skolemisation

A final and rather uncontroversial transformation carried out by `ACERules` we call "top level skolemisation". Constants (so called skolem constants) are introduced for existentially quantified variables that do not appear nested (i.e. are at the "top level").

Example 17. *Consider the ACE text*

```
There is a ball.
If Andrea passes the ball to Julia then Julia passes //
                                        the ball to Mary.
```

After the transformations detailed in sections 4.1.2 and 4.1.3 the `APE` *parse of this text is as follows:*

$$\begin{aligned}
&object(named(Mary), Mary, named, na, eq, 1)\wedge\\
&object(named(Julia), Julia, named, na, eq, 1)\wedge\\
&object(named(Andrea), Andrea, named, na, eq, 1)\wedge\\
&\exists A\Big((object(A, ball, countable, na, eq, 1)\wedge\\
&\quad pred\_mod(pass, named(Andrea), A, [modifier\_pp(to, named(Julia))]))\\
&\quad \rightarrow\\
&\quad group([pred\_mod(pass, named(Julia), A, [modifier\_pp(to, named(Mary))]]))\Big)
\end{aligned}$$

After top-level skolemisation the variable "A" is replaced with a constant "$v(0)$":

$$\begin{aligned}
&object(named(Mary), Mary, named, na, eq, 1)\wedge\\
&object(named(Julia), Julia, named, na, eq, 1)\wedge\\
&object(named(Andrea), Andrea, named, na, eq, 1)\wedge\\
&\Big((object(v(0), ball, countable, na, eq, 1)\wedge\\
&\quad pred\_mod(pass, named(Andrea), v(0), [modifier\_pp(to, named(Julia))]))\\
&\quad \rightarrow\\
&\quad group([pred\_mod(pass, named(Julia), v(0), [modifier\_pp(to, named(Mary))]]))\Big)
\end{aligned}$$

Thus, the text has been transformed into a format amenable to treatment by rule-systems to which `ACERules` *provides an interface to.*

4.2 A novel ACE interface to defeasible rules

We have in Section 4.1 given a detailed yet high level description of the main existing open source interface to rule systems for ACE, `ACERules`. As already indicated in the introduction to this chapter of our work (Section 4), reverse engineering such a description was motivated by problems we encountered when building a simple experimental prototype on top of `ACERules` to evaluate ACE texts extended with a construct for expressing defeasible rules via the direct stable semantics (Section 2.4). As part of the description of the system `ACERules` in Section 4.1 (see in particular sections 4.1.2.1, 4.1.2.2, 4.1.2.3, and 4.1.3) we have now pinpointed the sources of the issues we found.

We turn to presenting an alternative design for implementing the EMIL pipeline, built with the intention of overcoming some of the shortcomings we encountered with our prototype built on top of `ACERules` (Section 4.2.3). We first nevertheless briefly detail the minor modifications of the grammar of ACE as well as the system `APE` we carried out to allow expressing defeasible rules using ACE (Section 4.2.1). Also, we give a brief description (Section 4.2.2) of the prototype built on top of `ACERules`, dubbed `dACERules`, experience with which led to the system design we detail in what remains of this chapter.

4.2.1 Expressing defeasible rules in ACE

For adding means of expressing defeasible rules to ACE we focused on one of the most common ways of expressing defeasible rules that is also the form of defeasible implication considered when motivating the direct stable semantics [SW17, WS17]. This is defeasibility introduced via normality or typicality assumptions.

To add normality assumptions to ACE we made use of the grammatical infrastructure available in ACE to express modalities (such as "it is possible that" and "it is admissible that") in combination with complete sentences (see Section 3.4.4.4 of [ace13a]), i.e. as a form of subordination. Specifically, we have added the construct "it is usual that" as a further means of subordination. Thus, an example of an assumption written using our simple extension of ACE is the following:

```
It is usual that Mary wakes-up early.
```

Its negation, on the other hand, is written as follows:

```
It is not usual that Mary wakes-up early.
```

Examples of defeasible rules are:

```
If Mary is happy then it is usual that she smiles.
If X is a bird then it is usual that X flies.
If someone owns a house then it is usual that he/she owns a car.
```

To make APE "aware" of defeasible rules we have essentially added "usual" as a further modal operator to the list of modal operators in the source code of APE and extended the rules available for the modal operators to also encompass "usual". Thus, the modified version of APE returns DRSs with parses prefixed by a marker for defeasible rules for sentences subordinated to the fixed phrases "it is usual that" / "it is not usual that".

Note that we have not (yet) added support for constructs for "usual" analogous to modal auxiliaries that can be used with verb phrases (Section 2.3.5 of [ace13a]). E.g. APE identifies the use of "must" as a modal auxiliary in the sentence "John must wait" while our modification of APE does not identify the use of "usually" as an analogous auxiliary in "John usually waits". Hence, "usually" in the latter sentence is, as was already the case prior to our modifications, interpreted by APE as an adverb which modifies the verb "waiting".

4.2.2 A simple prototype on top of `ACERules`

As an initial testing-ground for the EMIL pipeline, we wrote a relatively simple script that mutates AceRules into a CNL interface to defeasible theories evaluated under the direct stable semantics. We reported on initial results in using this tool, dACERules, in [DWS17][18].

For our script we separated the AceRules parser and verbaliser components (which in turn, make use of the APE parser and verbalisation tools). The script then consists in a interleaving of calls to the AceRules (+ APE) parser, existing ASP encodings for the direct-stable semantics (from [SW17]) with an ASP solver (we used clingo), and finally the AceRules (+ APE) verbaliser. Crucially, we pre-process the input text removing all constructs indicating defeasibility and make use of the AceRules (+ APE) parser "as if" all rules in the input were strict, but at the same time tracking which rules are defeasible and which are not (for this we make use of labels that can be attached to rules when using the courteous semantics in AceRules).

By differentiating the rules in this way, we are able to use the afore-mentioned encodings for the direct-stable semantics (together with an ASP solver) later on in the pipeline. At the level of the stable sets, the distinction between strict and defeasible rules is irrelevant; and we are, hence, also able to make direct use of the AceRules (+ APE) verbaliser component. We will refer to the ASP encodings for the direct stable semantics we made use of at the back end of dACERules in some more detail in Section 4.2.4.2 as we also devise an alternative ASP encoding strategy for the revamped implementation of the EMIL pipeline we describe in Section 4.2.3.

[18]It is available online, packaged together with the source code of other systems it depends on, at https://www.dbai.tuwien.ac.at/proj/adf/dAceRules/

4.2.3 System design

We turn to describing our alternative (to the system `dACERules` described in Section 4.2.2) strategy for implementing the `EMIL` pipeline. The input to our novel system, which we dub `emil`, is as with `ACERules` (and `dACERules`) an ACE text. We parse the ACE text with the slightly modified version of `APE` described in Section 4.2.1. Just as `ACERules` (Section 4.1.1), we have an initial stage filtering DRSs amenable to further processing by our system.

It is also at this filtering stage that differences with `ACERules` emerge. Specifically, in contrast to `ACERules`, we disallow negation as failure as well as modality constructs (the interaction of these with constructs for expressing defeasibility remain to be investigated). On the other hand, `emil` obviously accepts DRSs containing sub-DRSs (sub-formulas) corresponding to assumptions and defeasible rules.

Modulo the mentioned constructs we nevertheless filter DRSs with exactly the same structure as `ACERules`, i.e. DRSs corresponding (after transformation of double implication; see Section 4.1.1) to collections of (now, defeasible as well as strict) rules having wide literals in the head as well as in the body (but no negation-as-failure). There is indeed a good reason for `ACERules` attempting to make such more general rules, rules with existentially quantified variables and wide literals or ∃-rules for short, conform to the more restricted rule format of the rule systems it provides an interface to. The reason being that rules expressed as ACE texts quite naturally correspond to ∃-rules, while very few of them seem to correspond to the more restricted “normal” rule format of e.g. ASP or defeasible programs[19].

The central difference of `emil` with `ACERules` is that `emil` accepts *all* DRSs corresponding to (strict and defeasible) ∃-rules, while `ACERules` ultimately only accepts those (strict) ∃-rules (with negation-as-failure) that can be made to conform through the several modifications detailed in Section 4.1. The central of these modifications are “predicate condensation” and especially “grouping” which as currently implemented in `ACERules` has a few significant shortcomings, as we detailed in Section 4.1.2.2 and Section 4.1.3.

We devise a scheme for compiling ∃-rules into normal rules that can be seen as a form of systematic meaning-preserving grouping for ∃-rules not having implicitly existentially quantified variables. Rules having existentially quantified variables on the other hand require a special treatment, which among other things requires making use of function symbols and, thus, the expressive power of full first order defeasible theories (which, given the possibility of infinite groundings can be expected to be, just as ASP programs with function symbols, undecidable in general). We describe our translation of ∃-rules into normal rules in Section 4.2.4.1. Moreover, we have an alternative encoding of the evaluation of defeasible theories via the direct stable semantics optimised to theories with

[19] Although the expressivity of the rules and experience with the system `ACERules` give some indication, to what extent rules expressed in natural language can be made to correspond to ∃-rules is ultimately an empirical question that remains to be investigated.

function symbols to ASP (also with function symbols) which we motivate and describe in Section 4.2.4.2.

We hence have an interface to a more expressive rule language which we compile into normal (strict and defeasible) rules (with function symbols) and, ultimately, normal ASP programs with function symbols. As a consequence, reasoning becomes undecidable in general but the rule language clearly subsumes normal (strict and defeasible) rules and, therefore, mechanisms in particular for avoiding existentially quantified variables such as e.g. grouping (improved to avoid the problems detailed in Section 4.1.3) can also be ported to this setting. On the other hand, our implementation strategy also allows the use of existentially quantified variables via function symbols (skolemisation) when such quantification cannot be avoided (as will often be the case in natural language texts; especially for expressing ontological knowledge). In particular, many subclasses of ASP programs with function symbols are known to be decidable (see e.g. [AZZ17]) and there are also solving strategies devised for ASP programs having infinite groundings (see e.g. [LBSG17]).

The implementation strategy behind `emil` also has the additional merit that several of the problems we identified in `ACERules` can be solved in a uniform manner. Specifically, as have already hinted at a form of predicate condensation and grouping are carried out in a uniform, systematic, and meaning-preserving manner via our rewriting of $\exists$ rules to normal rules. We describe the translation in question in Section 4.2.4.1.

On the other hand, further transformations of DRSs, such as are also incorporated in `ACERules`, either adding elements for facilitating natural language understanding or simplifying the logical form of DRSs (in particular, removing existentially quantified variables whenever possible) are desirable. Our current implementation is rather minimal in this regard; clearly many more optimisations in particular for facilitating natural language understanding (e.g. some of which, like inferences on the base of plural nouns, which are incorporated into `RACE` but not ACERules; see [Fuc10]) would be possible. To conclude our description of `emil` we detail our modifications as well as adoptions of elements of `ACERules` detailed in sections 4.1.2.1, 4.1.2.3, 4.1.2.4, and 4.1.4. Some of these are still planned, others already incorporated in our current implementation (see Section 4.4).

First of all, we do not follow `ACERules` in replacing predicates standing for indefinite pronouns with atoms standing for “anonymous objects”. As we argued in Section 4.1.2.1, this modification introduces errors. A solution which does not lead to errors is to leave the specification of which nouns refer to persons and which to inanimate objects to the user. Ideally, this ontological knowledge would be built into `emil` in the form of additional rules.

Regarding the processing of predicates standing for the copula “be”, we plan to carry out the transformations we detailed for `ACERules` (removing the existentially quantified variable introduced by the copula) only when the transformations do not lead to the errors we detailed in Section 4.1.2.3. I.e. for when the copula links proper name or noun phrase to noun phrase; also a proper name or noun phrase to an adjective. We will leave further uses of the copula underspecified for now. The latter is, in particular, the case for when the copula links a proper name or noun phrase to a noun phrase with an adjective.

Features of `ACERules` we plan to incorporate are transformation of double implications into normal implications (see Section 4.1.1). We also want to incorporate the strategy for condensation of predicates involving the preposition "of" from `ACERules` (thus removing the quantified variables introduced by the relation) as is (Section 4.1.2.4). In our current implementation we already add "object" atoms for named entities (Section 4.1.2.1) Finally, we likewise implement top-level skolemisation (Section 4.1.4)).

4.2.4 ASP based encodings for evaluating conflicting existential defeasible ACE rules

We turn now, first of all, to describe the more expressive rule language we use to represent ACE texts (whenever possible). We consider, especially, the translation of such ∃ - (strict and defeasible) rules into normal rules (with function symbols). This is the content of Section 4.2.4.1. In Section 4.2.4.2 we then motivate and detail a dynamic approach to encoding defeasible theories with function symbols to answer set programming (with function symbols). Both the translation of ∃-rules to normal rules as well as of defeasible theories to ASP programs is implemented in our system `emil`, whose functioning we describe in Section 4.4.

4.2.4.1 Expressing existential defeasible rules with wide literals as normal defeasible rules

As already indicated, the crucial difference between the system design outlined in Section 4.2.3 and that of `ACERules` (and `dACERules`) is that we support all ACE texts parsed to DRSs that correspond to collections of strict and defeasible rules having wide literals in the head and body (see Section 4.1.1). Concretely, such ∃-rules hence have the form

$$b_1, \dots, b_m, \neg(n^1_1, \dots, n^1_{u_1}), \dots, \neg(n^s_1, \dots, n^s_{u_s}) \rhd H$$

where e.g. $\neg(n^1_1, \dots, n^1_{u_1})$ is a "wide literal", $\rhd \in \{\rightarrow, \Rightarrow\}$, and H is of the form $h_1, \dots, h_t$ or $\neg(h_1, \dots, h_t)$. Also, $h_1, \dots, h_t$, $b_1, \dots, b_m$, $n^1_1, \dots, n^1_{u_1}, \dots, n^s_1, \dots, n^s_{u_s}$ are atoms as defined in Section 2.2.1 and $m, s \geq 0$, and $t, u_1, \dots, u_s \geq 1$. Variables occurring in the negative atoms $n^1_1, \dots, n^1_{u_1}, \dots, n^s_1, \dots, n^s_{u_s}$ but not in the positive atoms $b_1, \dots, b_m$ are interpreted as existentially quantified. The same holds for those variables occurring in the head H but not in the positive atoms in the body $b_1, \dots, b_m$.

We provide meaning to such ∃-rules via encodings to defeasible theories in a way that is quite similar to the encoding of ∃-ASP into ASP as defined by [GGLS15]. Namely, we have a "normalization" phase to remove the conjunctions of atoms from negative parts of the rules as well as remove existential variables from these negative parts. We also use skolemization to remove existential variables in the positive heads of rules. Moreover, we have an "expansion" phase to remove conjunctions of atoms in positive heads of the rules.

Crucial differences are due to the fact that our interpretation of negation is different to that of ∃-ASP and that we can make use of defeasible implication. In particular hence we allow negation over conjunctions of atoms in the heads of rules which we need to treat. We also need to consider defeasible rules. Furthermore, our "normalization" phase is different to that needed for ∃-ASP. Nevertheless, the technicalities are more or less standard, and there are sufficient similarities that it is relatively straightforward (although cumbersome) to translate the formal exposition of [GGLS15] to that of our scenario, so we rather give an example-based explanation of our translation here.

To start, let us consider the ACE rule "If someone owns a car then he/she owns a house"; i.e. an ACE rule corresponding to an ∃-rule having positive wide literals in the body as well as in the head. The result of parsing this sentence using `APE` is the rule

$$\begin{aligned}\forall A, B, C\Big(&(object(A, somebody, countable, na, eq, 1) \ \wedge \\ &object(B, car, countable, na, eq, 1) \wedge predicate(C, own, A, B)) \\ &\rightarrow \\ &\exists D, E\big(object(D, house, countable, na, eq, 1) \wedge predicate(E, own, A, D)\big)\Big)\end{aligned}$$

which, as an ∃-rule can be represented as follows:

$$\begin{gathered}object(A, somebody, countable, na, eq, 1), object(B, car, countable, na, eq, 1), \\ predicate(C, own, A, B) \rightarrow \\ object(D, house, countable, na, eq, 1), predicate(E, own, A, D)\end{gathered}$$

To ease the presentation in what follows we simplify the representation of the logical atoms in rules obtained from ACE texts by ignoring parameters which do not directly pertain to the logical form. For the rule in our example we thus have the following:

$$\begin{gathered}object(A, somebody), object(B, car), predicate(C, own, A, B) \\ \rightarrow \\ object(D, house), predicate(E, own, A, D)\end{gathered}$$

In our translation to normal rules we replace the atoms in the head of the rule with an auxiliary atom:

$$object(A, somebody), object(B, car), predicate(C, own, A, B) \rightarrow x\_auxPH1(A, B, C)$$

and add the rules

$$x\_auxPH1(A, B, C) \rightarrow object(x\_sk1(A, B, C), house)$$
$$x\_auxPH1(A, B, C) \rightarrow predicate(x\_sk2(A, B, C), own, A, x\_sk1(A, B, C))$$

encoding the meaning of the auxiliary predicate. Here the skolem function $x\_xk1(A, B, C)$ is used to represent the house owned (by someone A who is in the relation C "owns" with a car B). The skolem function $x\_sk2(A, B, C)$ is used to denote the owns relation (introduced by someone owning a house).

Note that replacing the head of the rule representing our example sentence ("if someone owns a car then he/she owns a house") by an auxiliary atom can be seen as a form of unrestricted grouping. Avoidance of the need to check for any further conditions for grouping to succeed (as is necessary in grouping as defined in `ACERules`; see Section 4.1.3) is obtained by the additional rules and the use of skolemisation. Also note that the skolem functions introduced by our translation are as "specific" (or "grounded") as the rules from which they are derived. As a further example, consider for instance the second sentence in the following snippet of discourse:

```
There is a ball.
If Andrea passes the ball to Julia then Julia passes //
                                          the ball to Mary.
```

which as an $\exists$ rule derived from the `APE` parse can be represented (in simplified form) as follows:

$$\begin{array}{c} object(B, ball), predicate(A, pass, andrea, B), modifier\_pp(A, to, julia) \\ \rightarrow \\ predicate(C, pass, julia, B), modifier\_pp(C, to, mary) \end{array}$$

After top-level skolemisation as is carried out in `ACERules` (Section 4.1.4) and which we also implement (see Section 4.2.3) such a rule gets translated into the rule

$$object(x\_c1, ball), predicate(A, pass, andrea, x\_c1), modifier\_pp(A, to, julia)$$

$$\rightarrow$$
$$x\_auxPH1(A, x\_c1)$$

with auxiliary rules

$$x\_auxPH1(A, x\_c1) \rightarrow predicate(x\_sk(A, x\_c1), pass, julia, x\_c1)$$
$$x\_auxPH1(A, x\_c1) \rightarrow modifier\_pp(x\_sk(A, x\_c1), to, mary)$$

Note in particular that the only remaining variable is for the event introduced by the "passing" action. The example also shows how our form of grouping subsumes a form of predicate condensation. Nevertheless, our treatment, in contrast to `ACERules` (see Section 4.1.2.2), preserves the Neo-Davidsonian reading of verbs and their modifiers. The reason is that rather than merging together verbs and their modifiers in a single logical atom, we keep atoms for verbs and their modifiers separate and link them via an auxiliary atom and making use of skolem functions. As a consequence, we also preserve the "diamond inference pattern" for verbs and their modifiers which is not preserved by `ACERules` (see, again, Section 4.1.2.2).

Turning to the translation of defeasible ∃-rules with positive wide literals to normal rules; the translation is analogous to that of strict rules but there is the option of making the rules for the auxiliary predicates introduced by our translation defeasible or not. The issue at hand is deciding the scope of the defeasible implication "⇒". Consider for instance the following assumption:

```
It is usual that a ferry that starts in Vienna //
                                    services Bratislava.
```

which written as an ∃-rule has the following parse:

$$\Rightarrow object(A, ferry), predicate(B, start, A),$$
$$modifier\_pp(B, in, vienna), predicate(C, service, A, bratislava)$$

In our translation the rule gets replaced by the unary assumption

$$\Rightarrow x\_auxPH1()$$

accompanied by the auxiliary rules

$$
\begin{aligned}
&x\_auxPH1() \rhd_1 object(x\_sk1(), ferry)\\
&x\_auxPH1() \rhd_2 predicate(x\_sk2(), start, x\_sk1())\\
&x\_auxPH1() \rhd_3 modifier\_pp(x\_sk2(), in, vienna)\\
&x\_auxPH1() \rhd_4 predicate(x\_sk3(), service, x\_sk1(), bratislava)
\end{aligned}
$$

the question being whether each of the $\rhd_i$ $(1 \leq i \leq 4)$ should be either $\rightarrow$ or $\Rightarrow$. Having $\rhd_i = \rightarrow$ for every i would amount to interpreting the scope of $\Rightarrow$ to be over the entire subordinate phrase “a ferry that starts in Vienna services Bratislava”. In particular, this reading would be incompatible with either of the facts “the ferry does not start in Vienna” or “the ferry does not service Bratislava”. I.e. from the text

```
It is usual that a ferry that starts in Vienna //
                                       services Bratislava.
The ferry does not start in Vienna.
```

it would not follow that “there is a ferry that services Bratislava” (via the direct stable semantics, see Section 2.4.2). On the other hand, interpreting $\rhd_i = \Rightarrow$ for every i would allow for the conclusion that “there is a ferry that services Bratislava”, although the ferry in question does not start in Vienna. Nevertheless, we note that having $\rhd_i = \Rightarrow$ for every i allows for breaking the connection between the atoms associated via skolem constants; therefore a more correct encoding introduces further auxiliary atoms standing for “there is a ferry that starts in Vienna” and “there is a ferry that services Bratislava” respectively:

$$
\begin{aligned}
&x\_auxPH1() \Rightarrow x\_auxPH2()\\
&x\_auxPH1() \Rightarrow x\_auxPH3()\\
&x\_auxPH1() \rightarrow object(x\_sk1(), ferry)\\
&x\_auxPH2() \rightarrow predicate(x\_sk2(), start, x\_sk1())\\
&x\_auxPH2() \rightarrow modifier\_pp(x\_sk2(), in, vienna)\\
&x\_auxPH3() \rightarrow predicate(x\_sk3(), service, x\_sk1(), bratislava)
\end{aligned}
$$

For pragmatic reasons in our current implementation of `emil` we have implemented the option amounting to $\rhd_i = \rightarrow$ for every i (also because this option makes the link to the verbalisation component of `APE` easier), although the alternative reading may be more desirable in practice.

Turning to ∃-rules with negative wide literals in the head, consider the sentence “if someone owns a car then he/she does not own a house”. As an ∃-rule obtained from the APE parse, this sentence is as follows:

$$\begin{gathered} object(A, somebody), object(B, car), predicate(C, own, A, B) \\ \rightarrow \\ \neg(object(D, house), predicate(E, own, A, D)) \end{gathered}$$

In our translation we once more replace the head of the rule with an auxiliary predicate:

$$\begin{gathered} object(A, somebody), object(B, car), predicate(C, own, A, B) \\ \rightarrow \\ x\_auxNH1(A, B, C) \end{gathered}$$

and add rules giving the meaning of the auxiliary predicate:

$$\begin{gathered} object(D, house), x\_auxNH1(A, B, C) \rightarrow \neg predicate(E, own, A, D) \\ predicate(E, own, A, D), x\_auxNH1(A, B, C) \rightarrow \neg object(D, house) \end{gathered}$$

If we need the first rule to be safe (i.e. all variables occurring in the head occur in the body) we can e.g. collect all variables that stand for verbs in a special atom `pName/1` and hence replace the rule with

$$object(D, house), x\_auxNH1(A, B, C), pName(E) \rightarrow \neg predicate(E, own, A, D)$$

Note also that if we have more than two atoms appearing negated in the head of a rule we need to apply the above illustrated translation recursively. E.g. for “if someone owns a car then he/she does not own a big house” we use the (safe variant of the) auxiliary rules:

$$\begin{gathered} object(D, house), x\_auxNH1(A, B, C) \rightarrow x\_auxNH2(A, B, C, D) \\ predicate(E, own, A, D), x\_auxNH2(A, B, C, D) \rightarrow \neg property(D, big) \end{gathered}$$

$$
\begin{aligned}
&property(D, big), x\_auxNH2(A, B, C, D), pName(E) \rightarrow \neg predicate(E, own, A, D)\\
&predicate(E, own, A, D), property(D, big) \rightarrow x\_auxPH1(A, D, E)\\
&x\_auxPH1(A, D, E), x\_auxNH1(A, B, C) \rightarrow \neg object(D, house)
\end{aligned}
$$

In contrast to the situation with rules with positive wide literals in the head, for rules with negative wide literals in the head the treatment of strict and defeasible rules is exactly analogous. The reason is that ⇒ inherits the scope from ¬ in this case. I.e. the only difference in the translation of e.g. "if someone owns a car then it is usual that he/she does not own a house" w.r.t. the corresponding strict rule is that the main rule

$$
\begin{aligned}
&object(A, somebody), object(B, car), predicate(C, own, A, B)\\
&\qquad\Rightarrow\\
&x\_auxNH1(A, B, C)
\end{aligned}
$$

is defeasible, while the auxiliary rules defining `x_auxNH1/3` remain the same.

We consider now the conceptually slightly more intricate case in which negative wide literals appear in the heads of rules, e.g. for the sentence "if someone does not own a car then he/she owns a house". Here there are several options. The most straightforward (which we call option "E"), following more or less the treatment in [GGLS15], is to put the burden of proof on the existential assertion; i.e. by default no one owns a car. This option, which is the one implemented in the current version of `emil` (Section 4.4), can be encoded as follows (omitting the auxiliary rules for `x_auxPH1/1`) :

$$object(A, somebody), \neg x\_auxPB1(A) \rightarrow x\_auxPH1(A)$$

$$
\begin{aligned}
&object(B, car), predicate(C, own, A, B) \rightarrow x\_auxPB1(A)\\
&object(A, somebody) \Rightarrow \neg x\_auxPB1(A)
\end{aligned}
$$

An alternative is rather to put the burden of proof on the negation of the existential assertion, i.e. by default everyone owns a car (option "N"). We give here an encoding where all rules are safe (we use an additional atom `oName/1` for collecting constants standing for objects):

$$object(A, somebody), \neg x\_auxPB1(A) \rightarrow x\_auxPH1(A)$$

$$
\begin{aligned}
&\neg object(B, car), \neg predicate(C, own, A, B) \rightarrow x\_auxVNPB1(A, B, C)\\
&\neg object(B, car), predicate(C, own, A, B) \rightarrow x\_auxVNPB1(A, B, C)\\
&object(B, car), \neg predicate(C, own, A, B) \rightarrow x\_auxVNPB1(A, B, C)\\
&object(A, somebody), oName(B), pName(C) \Rightarrow \neg x\_auxVNPB1(A, B, C)\\
&\neg x\_auxVNPB1(A, B, C) \rightarrow x\_auxPB1(A)\\
&object(A, somebody) \Rightarrow \neg x\_auxPB1(A)
\end{aligned}
$$

The atom `x_auxVNPB1/3` is used here to "verify" for any "somebody" if the "somebody" in question owns a car. By default this is not verified.

Finally, several combinations of options "E" and "N" are conceivable. In particular, there is the safest but rather inefficient option to reason by cases, i.e. consider for everyone both the possibility that the person in question owns a car as well as that he/she does not own a car. One simple encoding of this option would be as follows:

$$
object(A, somebody), \neg x\_auxPB1(A) \rightarrow x\_auxPH1(A)
$$

$$
\begin{aligned}
&object(A, somebody) \Rightarrow \neg x\_auxPB1(A)\\
&object(A, somebody) \Rightarrow x\_auxPB1(A)\\
&x\_auxPB1(A) \rightarrow object(x\_sk1(A), house)\\
&x\_auxPB1(A) \rightarrow predicate(x\_sk2(A), own, A, x\_sk1(A))\\
&\neg x\_auxPB1(A), predicate(C, own, A, B) \rightarrow \neg object(B, house)\\
&\neg x\_auxPB1(A), object(B, house), pName(C) \rightarrow \neg predicate(C, own, A, B)
\end{aligned}
$$

4.2.4.2 Dynamic ASP encoding for the direct stable semantics

As we already hinted at in Section 4.2.2, encodings to ASP for evaluating defeasible theories via the direct stable semantics are reported on in [SW17] and publicly available[20]. We have made use of these encodings in the prototype `dACERules`. We will call these encodings to ASP "Strass's encodings". The encodings in question are static in the sense that only the part of the encoding used for specifying the defeasible theory changes with the input. The module encoding the semantic evaluation of defeasible theories uses ASP-disjunction and remains fixed. Maximisation aspects (maximisation of defeasible rules to be applied in a consistency preserving manner) are implemented using the saturation technique [EG95] we also made use of in the encodings for ADFs presented in Section 3.2.3.

For specifying input defeasible theories to Strass's encodings, its constituent rules are represented by ASP terms. The binary predicates `head/2` and `body/2` declare rule

[20]At `https://github.com/hstrass/defeasible-rules`.

heads and bodies, respectively; predicate `def/1` declares a rule to be defeasible. We give an example of the specification of a variant of the classic “Tweety example” from [Rei78] as a defeasible theory in Example 18.

Example 18. *The following is a variant of the Tweety example specified as input to Strass's encodings for evaluating defeasible theories. Comments preceded by “%” detail the rules being codified.*

```
% constants
c(tweety).
c(tux).

%flies(C) <= bird(C)
def(bf(C)) :- c(C).
body(bf(C), bird(C)) :- c(C).
head(bf(C), flies(C)) :- c(C).

%flies(C) <- penguin(C)
body(pn(C), penguin(C)) :- c(C).
head(pn(C), neg(flies(C))) :- c(C).

%bird(tweety) <-
head(twb, bird(tweety)).
% penguin(tux) <-
head(tup, penguin(tux)).
```

Note, in particular, the use of the predicate `c/1` *for grounding the rules w.r.t. the constants (*`tweety` *and* `tux`*) in the program.*

Strass's encodings have the agreeable property that, while also working for first order defeasible theories, they are complexity sensitive for propositional defeasible theories; this being because of the matching complexity of reasoning for propositional defeasible theories and the data complexity of disjunctive ASP (the existence problem for both formalisms is Σ_2^P-complete; see sections 2.2.4 and 2.4.3). Also, the encoding reflects the definition of the semantics of first order defeasible theories in terms of separate grounding and evaluation phases.

The latter nevertheless also points to a potential drawback of Strass's encodings for practical purposes; the problem being that, to preserve the distinction between the data and the program, defeasible theories need to be specified essentially as facts and hence the grounding needs to be generated explicitly (in most cases) while this is usually not the case for first order ASP programs. Consider, for instance, the strict rule

$$q(X), p(X_1), \ldots, p(X_n) \rightarrow p(X)$$

as an ASP rule, i.e.,

$$p(X) \text{ :- } q(X), p(X_1), ..., p(X_n).$$

and assume it needs to be grounded for two constants a and b. Naive grounding would yield 2^n ground instances of the rule (since for each X_i the grounding procedure can choose between a and b), while most ASP grounders will realize that this rule actually amounts to the rules [BET11]

$$\begin{array}{c} p(a) \text{ :- } q(a), p(b). \\ p(b) \text{ :- } q(b), p(a). \end{array}$$

whatever the size of n. Now consider the specification of the aforementioned rule in the context of Strass's encodings:

```
c(a). c(b).

head(p(X,X1,...,XN), p(X)) :- c(X),c(X1),...,c(XN).

body(p(X,X1,...,XN),q(X)) :- c(X),c(X1),...,c(XN).

body(p(X,X1,...,XN),p(X1)) :- c(X),c(X1),...,c(XN).

...

body(p(X,X1,...,XN),p(XN)) :- c(X),c(X_1),...,c(XN).
```

Here the structure of the original rule is broken and thus, for instance, the ASP solver `clingo` (version 5.3.0) times out when using Strass's encodings on this example together with some facts ($q(a)$, $q(b)$, $p(b)$) for $n = 16$[21], while is able to solve the equivalent ASP version under 1 second (0.008 seconds).

The shortcomings of Strass's encodings for defeasible theories with variables are particularly evident when using skolemisation (as we need to make use of for the translation from

[21]Time out of ten minutes. On a 4 GB openSUSE (42.3) machine with 4 Intel Core processors (3.30 GHz)

defeasible theories with existentially quantified variables to normal defeasible theories; see Section 4.2.4.1). Consider for instance the simple ASP program with the function symbol `f` and constant `a`:

$$\begin{aligned} o(a)&. \\ p(f(X)) \text{ :- } o(X)&. \\ q(f(X)) \text{ :- } p(X)&. \end{aligned}$$

which has the unique answer set $\{a, p(f(a)), q(f(a))\}$, computed in under one second by `clingo`. In order to be evaluated using Strass's encodings we need to specify the ASP program as follows:

```
c(a).
c(f(X)):-c(X).


head(r0,o(a)).

head(r1(X),p(f(X))):- c(X).
body(r1(X),o(X)):-c(X).

head(r2(X),q(f(X))):-c(X).
body(r2(X),p(X)):-c(X).
```

Note in particular the need of the rule

```
c(f(X)):-c(X).
```

for declaring all possible applications of the function symbols f as constants. When feeding the program in question together with Strass's encodings e.g. to `clingo` there are memory errors (”std::bad_alloc”) after 48.566 seconds computing time.

In conclusion, Strass's encodings for defeasible theories have the drawback that the constituents of rules need to be specified essentially as facts. This breaks the structure of the rules, frustrating built in strategies of ASP grounders to avoid an exponential or even infinite blowup when grounding. This is not to say that a more intelligent grounding mechanism could not also be devised for defeasible theories, rather that this would basically amount to re-implementing the grounding mechanisms of a specific ASP-solver, while encodings that preserve the structure of the rules in an input defeasible theory would allow us to piggyback on the grounding developments for any ASP grounder (+solver) we wish to experiment with. As we have seen, this is particularly desirable when making use of skolemisation, as naive grounding easily leads to infinite programs in this case.

For the mentioned reason we sketch in what follows a dynamic (i.e. both the data and program change with the input), yet structure-preserving alternative to Strass's encodings for defeasible theories with variables. Moreover, the encoding is to non-disjunctive ASP, hence also indicating that, contrasting with the results for propositional theories (see Section 2.4.3), the direct stable semantics for first order defeasible theories is not more expressive than normal (i.e. non disjunctive) ASP with variables. In particular, the encoding does not make use of the saturation technique and thus is, arguably, also simpler than Strass's encoding.

Our dynamic encoding of the evaluation of defeasible theories to ASP works by first guessing what defeasible rules to apply (for each possible ground instance of the rules) and then checking whether more defeasible rules could have been applied without making the program inconsistent. It filters out those guesses for which the latter does not hold (i.e. applying more defeasible rules would make the program inconsistent). We present our encoding by example, showing the encoding of the evaluation of the Tweety example (Example 18) as an ASP program.

As indicated, the encoding starts with an initial guess of the defeasible rules to apply. All strict rules must be applied. The predicate `holds/1` is used to compute the closure of the defeasible theory w.r.t. all of the strict rules and the guess of defeasible rules. In the Tweety example there is only one defeasible rule ($flies(X) \Leftarrow bird(X)$); the following ASP rules thus suffice for encoding the choice of applying each possible ground instance of the rule (labelled using the constant `r1`) for which the body of the rule holds (the latter for avoiding unnecessary guesses):

```
apply(r1,X,y) :- not apply(r1,X,n),holds(bird(X)).
apply(r1,X,n) :- not apply(r1,X,y),holds(bird(X)).
```

The following ASP rules then compute the closure w.r.t. the strict rules and guess of defeasible rules for the Tweety example:

```
holds(bird(tweety)).
holds(penguin(tux)).
holds(neg(flies(X))) :- holds(penguin(X)).
holds(bird(X)) :- holds(penguin(X)).
holds(flies(X)) :- apply(r1,X,y).
```

Note, in particular, that `holds(flies(X))` is derived only for a ground substitution of X for which the defeasible rule $r1$ is applied. To represent a stable set, the literals that result from computing the closure w.r.t. the strict rules and guessed defeasible rules, needs to be consistent. To check this our encoding uses a rule codifying consistency (separate, because we need to check consistency also in other parts of the encoding) and a constraint amounting to disallowing guesses of defeasible rules that result in an inconsistent set of literals.

```
inconsistent :- holds(neg(X)),holds(X).
:- inconsistent.
```

We have, until now, shown how to codify a guess for defeasible rules to apply and the computation of the closure w.r.t. the strict rules and guessed defeasible rules. Moreover, our encoding disallows guesses of defeasible rules for which the closure is inconsistent. Note also that the computation of the closure via ASP rules mirroring the strict and defeasible rules guarantees that there is a derivation (as required by the definition of the direct stable semantics; see Section 2.4.2) for each literal in the closure w.r.t. the strict and defeasible rules in question. The reason is (the acyclic nature of) the order in which ASP rules are applied to compute the minimal models of the reducts of programs w.r.t. an ASP interpretation.

We now show how to also encode that the guessed defeasible rules to apply are maximal w.r.t. consistency; i.e. that there is no ground instance of a defeasible rule that has not been applied but could also have been applied while preserving consistency of the closure w.r.t. the resulting rules. For this purpose we use a predicate *holds*/3 (note the arity 3), to store the result of computing the closure w.r.t. any possible extension of the initial guess of defeasible rules by exactly one ground instance of a defeasible rule. We thus again need to duplicate the input defeasible theory in our ASP encoding; for the Tweety example this results in the following rules:

```
holds(r1,Z,bird(tweety)) :- apply(r1,Z,n).
holds(r1,Z,penguin(tux)) :- apply(r1,Z,n).
holds(r1,Z,neg(flies(X))) :- holds(r1,Z,penguin(X)),//
                                                apply(r1,Z,n).
holds(r1,Z,bird(X)) :- holds(r1,Z,penguin(X)),apply(r1,Z,n).
holds(r1,Z,flies(X)) :- apply(r1,X,y),apply(r1,Z,n).
holds(r1,Z,flies(Z)) :- apply(r1,Z,n).
```

The parameters `r1` and `Z` of the `holds/3` predicate here indicate that the closure is being computed w.r.t. a particular extension of the initial guess of defeasible rules to apply; i.e. the initial guess is expanded to also include the rule $r1$ for a specific ground substitution of the variables in the body of $r1$ (indicated by Z). The following rules, which complete the encoding, now disallow an initial guess for which there is a ground instance of a defeasible rule (in our example $r1$) which could have been applied in a consistency preserving manner:

```
inconsistent(r1,Z) :- holds(r1,Z,neg(X)),holds(r1,Z,X).
:- not inconsistent(r1,Z),apply(r1,Z,n).
```

Example 19 shows all of our dynamic encoding for Example 18, yet written in a more succinct manner. In the encoding we use use identifiers for guesses of defeasible rules to apply (wrapped in the predicate `relrid/1`) and, hence, can merge the computation of the closure as well as check for consistency w.r.t. the different guesses.

Example 19. *The following shows a succinct version of our dynamic ASP encoding for Example 18 with comments preceded by "%" to ease understanding.*

```
%Identifiers for relevant guesses of defeasible rules
%"b" identifies the "base guess"

relrid(b).
relrid(I):- apply(I,n).

%Bodies and heads of defeasible rules

hbody(I,rid(r1,args(X))):-holds(I,bird(X)).
holds(I,flies(X)):-hhead(I,rid(r1,args(X))).


%The base guess:
%Choose whether to apply or not apply
%the defeasible rule + input identified by D

apply(D,y) :- not apply(D,n),hbody(b,D).
apply(D,n) :- not apply(D,y),hbody(b,D).

%Duplication of rules to compute closure
%w.r.t relevant guesses of defeasible (+ strict) rules

holds(I,bird(tweety)) :- relrid(I).
holds(I,penguin(tux)) :- relrid(I).
holds(I,neg(flies(X))) :- holds(I,penguin(X)),relrid(I).
holds(I,bird(X)) :- holds(I,penguin(X)),relrid(I).


hhead(I,D) :- apply(D,y),relrid(I).
hhead(I,I) :- relrid(I),I!=b.

%Closure w.r.t. guess with identiifer I is inconsistent
```

```
inconsistent(I):- holds(I,neg(X)),holds(I,X).

%Constraints

:- inconsistent(b).
:- not inconsistent(I),relrid(I),I!=b.

%Output
holds(X):- holds(b,X).
```

4.3 An extended example

In this section we use our implementation of the EMIL pipeline to motivate the latter. The results we refer to in this section have been obtained with the system described in Section 4.2.2. We stick to the version that we report on in [DWS17] mainly because our current implementation of `emil` still leaves the copula "be" underspecified and the interaction with the verbalisation component of `APE` also still needs to be improved. In any case, we make some comments about parts of the example that can be simplified by using our current newer system (see also further discussion in Section 4.4).

We motivate EMIL in the context of AceWiki [Kuh09],[22] a prototype of an encyclopedia in the style of the popular Wikipedia,[23] but where articles are written using ACE rather than unrestricted natural language. The advantage to using ACE in a wiki is that non-expert users can edit AceWiki entries, while at the same time users can use complex question answering and draw inferences. As it currently stands, AceWiki can represent a consistent KB about some domain and uses only strict rules. We base our example on current entries in the AceWiki about geographical information,[24] which have been restricted to a variant of ACE that can be translated into the rule language OWL 2 RL[GHVD03] and thus also, in principle, into the fragment of ACE admitted by EMIL.

As a motivating example, thus consider the entry for *island* in the geographical AceWiki. Some straight-forward statements pertaining to the definition of *island* appear, e.g. *Every island is a land-mass* and *Every island is surrounded by a body of water.* Using ACE, such statements can be written in a straightforward manner:

```
(1) Every island is a land-mass.
(2) If X is an island then a body-of-water surrounds X.
```

As already indicated in Section 4.0.1 `APE` enables the addition of lexical entries, such as proper names *Mainland-Shetland* or *St-Ninians-Isle.* Moreover, `APE` is often able to

[22]AceWiki can be accessed at `http://attempto.ifi.uzh.ch/acewiki/`.
[23]`https://www.wikipedia.org/`
[24]`http://attempto.ifi.uzh.ch/webapps/acewikigeo/`

deduce the word class for words that are not in its lexicon from the context. There are some interactions in `APE/AceRules` in relation to the verb form, quantifier scope, and the verbaliser (among other subtleties) such that, for example, we have represented (2) as a rule; we suppress further such incidental comments. We do however further note that rule (2) illustrates the situation where the input text introduces an implicitly existentially quantified variable in the head of a rule (here, referring to a body-of-water).

The problem, which we develop, is to add a new entry for *tied-island* to this AceWiki. However, as we show, this would lead to inconsistency were we to only have strict rules. According to Wikipedia, tied islands "are landforms consisting of an island that is connected to land only by a tombolo: a spit of beach materials connected to land at both ends."[25] With slight simplification, this definition can be written into AceWiki as follows:

```
(3) Every tied-island is an island.
(4) Every tied-island attaches-to a land-mass.
```

A prominent example of a *tied-island* according to the Wikipedia entry is *St. Ninian's Isle*, which is attached to *Mainland Shetland*, the largest of the Shetland Islands off the coast of Scotland. Thus, entries for St. Ninian's Isle and Mainland Shetland in AceWiki would be:

```
(5) Mainland-Shetland is an island.
(6) St-Ninians-Isle is a tied-island.
(7) St-Ninians-Isle is a part of the Shetland-Islands.
```

According to the Wikipedia entry for St. Ninian's Isle, during the winter strong wave action removes sand from the tombolo that connects St. Ninian to Mainland Shetland such that the tombolo is usually covered at high tide and occasionally throughout the tidal cycle. Hence, simply stating that *St. Ninian's Isle attaches to Mainland Shetland* would be incorrect. Spelling out the exact conditions under which St. Ninian's Isle is connected to Mainland-Shetland, which corresponds to using exceptions in strict rules, seems quite difficult if even possible (or desirable) and would be rather uncommon for an application like AceWiki. Rather, an easy solution is provided by the use of the predicate *it is usual that* applied to a statement:

```
(8) It is usual that St-Ninians-Isle attaches-to //
                                         Mainland-Shetland.
```

Let us now turn to a more fundamental reason for being able to distinguish between defeasible and strict statements in a CNL. Consider now the result of having all of

[25]`https://en.wikipedia.org/wiki/Tied_island` (accessed on 4/4/2017)

the above statements in the AceWiki together with the following fairly uncontroversial statements referring to the meanings of *being attached to a land mass*, *being surrounded by water*, and *being a part of.*

```
(9) If X attaches-to a land-mass then it is false that a //
    body-of-water surrounds X.
(10) If a body-of-water surrounds X then it is false that X
     attaches-to a land-mass.
(11) If St-Ninians-Isle attaches-to Mainland-Shetland then //
     St-Ninians-Isle is a part of Mainland-Shetland.
(12) If St-Ninians-Isle attaches-to Mainland-Shetland then //
     St-Ninians-Isle attaches-to a land-mass.
(13) If St-Ninians-Isle attaches-to a land-mass then //
     St-Ninians-Isle attaches-to Mainland-Shetland.
```

Here we note that we retrospectively know that the statements (12) and (13) were needed in our example because the grouping mechanism in `ACERules` (see Section 4.1.3) hinders our system `dACERules` to derive that "St Ninian's Isle attaches-to a land-mass" from "St Ninian's Isle attaches-to Mainland-Shetland"; in our current system (see Section4.4) these rules are not needed.

Since according to (6) St. Ninian's Isle is a tied island, and according to (3) every tied island is an island, and both (3) as well as (6) are strict rules, the direct stable semantics forces one to conclude that St. Ninian's Isle is an island. Now, because St. Ninian's Isle is an island and following (2), we conclude that a body of water surrounds St. Ninian's Isle. But from the fact that St. Ninian's is also a tied island and (4), St. Ninian's Isle attaches to a land mass. This leads to a contradiction according to statements (9) and (10). Hence, the entire AceWiki is deemed inconsistent and further reasoning is invalidated.

Note that the AceWiki remains inconsistent even after removing statement (8); the reason for the apparent contradiction in the Wiki is the fact, as is stated in the Wikipedia entry referring to St. Ninian's Isle,[26] that "[d]epending on the definition used, St. Ninian's is [...] either an island, or a peninsula." This reveals that the definition for *tied-island* in (3) should also be *defeasible.* However, in contrast to the reasons for the defeasiblity of (8), this is now due to the fact that there is no consensus on the meaning of *tied island.* Thus, we replace (3) with the more accurate statement:

```
(3′) If X is a tied-island then it is usual that X is an island.
```

The consequence is that there is now one stable set:

[26] `https://en.wikipedia.org/wiki/St_Ninian's_Isle` (accessed on 4.4.2017)

```
ANSWER-TEXT #1:

There is a body-of-water X1.
St-Ninians-Isle is a tied-island.
Mainland-Shetland is a land-mass.
Mainland-Shetland is an island.
St-Ninians-Isle is a part of Shetland-Islands.
St-Ninians-Isle is a part of Mainland-Shetland.
St-Ninians-Isle attaches-to a land-mass.
The body-of-water X1 surrounds Mainland-Shetland.
St-Ninians-Isle attaches-to Mainland-Shetland.
It is false that Mainland-Shetland attaches-to a land-mass.
It is false that a body-of-water surrounds St-Ninians-Isle.
```

Here the conclusion is that St. Ninian's Isle is a tied island that is attached to Mainland Shetland, while nothing can be said regarding whether St. Ninian's is also an island or not. The reason is that since statement (4) is strict, (8) is also effectively interpreted as a strict rule; that is, (8) strictly holds. To make (4) consistent with the *intended reading* of (8), (4) should be replaced with:

```
(4') If X is a tied-island then it is usual that X attaches-to a
     land-mass.
```

The result is that there are now two stable sets (answer-texts), which have in common the statements:

```
There is a body-of-water X1.
St-Ninians-Isle is a tied-island.
Mainland-Shetland is a land-mass.
Mainland-Shetland is an island.
St-Ninians-Isle is a part of Shetland-Islands.
The body-of-water X1 surrounds Mainland-Shetland.
It is false that Mainland-Shetland attaches-to a land-mass.
```

One stable set contains the following statements in addition to the common statements:

```
St-Ninians-Isle is a part of Mainland-Shetland.
St-Ninians-Isle attaches-to a land-mass.
St-Ninians-Isle attaches-to Mainland-Shetland.
It is false that a body-of-water surrounds St-Ninians-Isle.
```

The other stable set contains the following statements in addition to the common statements:

```
There is a body-of-water X2.
St-Ninians-Isle is a land-mass.
St-Ninians-Isle is an island.
The body-of-water X2 surrounds St-Ninians-Isle.
It is false that St-Ninians-Isle attaches-to a land-mass.
```

The interpretation of the latter set of statements is that St. Ninian's Isle is a tied island, but can only be called an island when it is not attached to Mainland-Shetland. Also relaxing the definition of *island* by changing (2) to

```
(2′) If X is an island then it is usual that a body-of-water
     surrounds X.
```

has the consequence that St. Ninian's Isle can also (always) be considered an island, despite the fact that the isle is not always surrounded by water.

Summarizing, we have shown that by distinguishing between defeasible and strict statements, we can resolve apparent inconsistencies such as might arise, in our example, because of the use of generic statements that allow for exceptions or because different meanings can be attached to certain words.

However, the EMIL pipeline does not require explicit statement of exceptions or alternatives. Using non-artificial, specific exceptions together with negation-as-failure in strict rules is often not feasible nor desirable. More fundamentally, using artificial exceptions, e.g. *abnormality* predicates specific to each rule, will usually not lead to a satisfactory result. Consider, for instance the effect of having the statement (8") below rather than the statement (8) mentioned previously, while replacing (3) with (3") rather than (3'), (4) with (4") rather than (4'), as well as (2) with (2") rather than (2').

```
(8′′)If it is not provable that it is false that
     St-Ninians-Isle attaches-to Mainland-Shetland //
     then St-Ninians-Isle attaches-to Mainland-Shetland.
(3′′) If X is a tied-island and it is not provable that //
      X is not an island then X is an island.
(4′′) If X is a tied-island and it is not provable that //
      it is false that X attaches-to a land-mass then X //
      attaches-to a land-mass.
(2′′) If X is an island and it is not provable that it //
      is false that a body-of-water surrounds X then a //
      body-of-water surrounds X.
```

The resulting text does not have any answer set under the standard stable semantics for logic programs.[27] Interpreting the text under the courteous semantics is also unsatisfactory in general. First, because the rules must be acyclic and second because the resulting answer is often uninformative or somewhat arbitrary. In the current case, the rules are in fact cyclic and hence no answer is produced.

4.4 Implementation and experiments

We have implemented the rudiments of the system `emil` as described in Section 4.2. Specifically, we implemented the filtering of ACE texts corresponding to (strict and defeasible) ∃-rules and the translation of such ∃-rules to normal defeasible rules as described in Section 4.2.4.2. We also implemented the ASP encodings to evaluate the defeasible rules via the direct-stable-semantics sketched in Section 4.2.4.1 and the translation of answer-sets (generated by running the answer-set solver `clingo`[28]) into DRSs that can be verbalised by `APE`. We also implemented some basic features such as top-level skolemisation (Section 4.1.4) also present in `ACERules`. An upcoming version of `emil` will include a corrected variant of processing of atoms corresponding to the copula “be” as described at the end of Section 4.2 as well as the treatment of atoms corresponding to the preposition “of” as is present in `ACERules`[29]. I.e., in the current available version of `emil` the copula “be” and preposition “of” are yet underspecified.

Our system is publicly available[30] and, as the other systems presented as part of this work, implemented in `Scala`[31] and can hence be run as a `Java`[32] executable. A typical call of `emil` (using a `UNIX` command line) looks as follows:

```
java -jar emil_0.0.0.jar -inFile infile -outFile outfile
```

Here “infile” is the name of the input file containing an ACE text and “outfile” is the file where the output of `emil` will be printed. The system assumes `SWI Prolog`[33] executables “./ape/ape.exe” which generates a DRS from an ACE text and “ape/apev.exe” which verbalises DRSs (files and instructions on how to generate such executables are on the webpage dedicated to `emil`). It also assumes the ASP solver `clingo` (tested with

[27]This is not to say that it is not possible to simulate the evaluation of ACE texts under the direct-stable semantics by using logic programs without the defeasible conditional; in fact the encodings referred to in sections 4.2.4.2 provide such a simulation. On the other hand, the complexity results from Section 2.4 also suggest that any such simulation via normal or extended logic programs will involve a worst-case exponential blow-up in general (unless the polynomial hierarchy collapses to its first level), at least for ACE texts which can be parsed as *grounded* defeasible theories.

[28]`https://potassco.org/clingo/`

[29]We have implemented this version (`emil` 0.0.1), but it remains to be more extensively tested.

[30]`https://www.dbai.tuwien.ac.at/proj/grappa/emil/`

[31]`https://www.scala-lang.org/`

[32]`https://www.java.com`

[33]`http://www.swi-prolog.org/`

version 4.5.4) is installed (and on the PATH environment variable) as well as access to the "/tmp" folder (for generating temporary files).

We provide the complete usage (subject to change in future versions) of `emil`:

```
usage: emil -inFile inFileName -outFile outFileName [options]
with options:
-h              display this help
-version        print version
-asperix        use asperix instead of clingo
-e              add some extra rules and ground rules for \
                 copula "be"
-d              print debug info
warning: copula "be" and preposition "of" underspecified when \
not using option e; option e favours intersective reading of \
copula and blurs distinction between inanimate and animate \
objects
```

Here option "-asperix" uses the ASP solver `ASPeRiX`[34] rather than `clingo` (yet not extensively tested) and option "-d" shows details of the pipeline of the implementation. The option "-e" adds some of the functionality of `ACERules` to our system. Specifically, it adds extra rules for making all named entities in the program instances of "somebodies" as well as all entities instances of "somethings" (see discussion in Section 4.1.2.1). While this leads to incorrect inferences in general (as argued in Section 4.1.2.1), it is still useful for some texts. Also, option "-e" specifies the meaning of the copula "be" by grounding variables introduced by the copula. E.g. the rule corresponding to the sentence

```
Every child is happy.
```

gets replaced with rules representing

```
If Mary is a child then he/she is happy.
If John is a child then he/she is happy.
```

for an ACE text where the only entities referred to in the input ACE text are "Mary" and "John". Just as the treatment of the copula "be" by `ACERules`, this favours an intersective reading of adjectives (see Section 4.1.2.3), but is useful for ACE texts where this assumption holds. This is a remnant of a previous iteration of the implementation which is still useful for some texts so we have decided to keep this option for running `emil` until we make available the version of `emil` which implements treatment of the copula "be" as detailed in Section 4.2.

[34] `http://www.info.univ-angers.fr/pub/claire/asperix/`

We have been able to successfully run our system `emil` on most test-cases (with some modifications in case of there being negation as failure as well as priorities over rules) available for `AceRules`. Concretely, we have been able to run `emil` on each of the 44 test cases in under one minute with option "e" and without. While without option "e" `emil` returns incorrect answers on most texts having the copula "be" (as is to be expected), with option "e" there are only errors (as far as we are able to tell) for 6 texts; all of them also related to the treatment of the copula "be". Otherwise, the interface with `APE`'s verbalisation component needs to be improved (i.e. some atoms are not verbalised at the moment); we note this is also something that `ACERules` has some issues with.

To conclude this section in Example 20 we give an example of the output of `emil`. Further examples on which to run `emil` (including the extended example from Section 4.3) can be found on the webpage dedicated to the system.

Example 20. *For the input text*

```
John drives in the countryside.
Mary drives in the countryside.
Suzy drives in the countryside.
There is a red car.
If John drives in the countryside then it is usual /
                           that John drives in the red car.
If Mary drives in the countryside then it is usual /
                           that Mary drives in the red car.
If Suzy drives in the countryside then it is usual /
                           that Suzy drives in the red car.
If John drives in the red car and Mary drives in the red car /
                       then Suzy does not drive in the red car.
If John drives in the red car and Suzy drives in the red car /
                       then Mary does not drive in the red car.
If Suzy drives in the red car and Mary drives in the red car /
                       then John does not drive in the red car.
```

the output of `emil` *(both with and without option "e") is:*

```
Answer-text #1:

Suzy drives in a red car X1.
Mary drives in the red car X1.
Suzy drives in a countryside X2.
Mary drives in the countryside X2.
John drives in the countryside X2.
```

```
Answer-text #2:

Mary drives in a red car X1.
John drives in the red car X1.
Suzy drives in a countryside X2.
Mary drives in the countryside X2.
John drives in the countryside X2.

Answer-text #3:

Suzy drives in a red car X1.
John drives in the red car X1.
Suzy drives in a countryside X2.
Mary drives in the countryside X2.
John drives in the countryside X2.
```

CHAPTER 5

Conclusion

We conclude this work by summarising, once more, our main contributions. Then we give some pointers on related and future work.

Summary In this study we presented realisation strategies for hard computational problems in abstract and structured argumentation based on reductions to quantified boolean formulas and answer-set programming. Specifically, we first of all developed QBF and ASP encodings for one of the most comprehensive current formalisms for abstract argumentation, ADFs [BES$^+$13]. More to the point, we started by presenting complexity-sensitive QBF encodings for verification as well as (credulous and skeptical) acceptance problems for the stable semantics (Section 3.1.2). Then (Section 3.1.3) we developed link-information-sensitive QBF encodings for the same reasoning problems for all main semantics for ADFs (admissible, complete, preferred, grounded, stable). The motivation behind the latter QBF encodings is that QBF solvers are able to latch on to the information about the link types and that this boost the performance of such solvers to the same degree that the reasoning tasks may become easier when the input ADFs are "close to" being bipolar (see [Wal14, SW15] and Section 3.1.3.1).

We also developed dynamic ASP encodings for acceptance problems w.r.t. the main semantics for ADFs (again: admissible, complete, preferred, grounded, stable) making use of the fact that the combined complexity of ASP programs with predicates of bounded arity, as is the case for the complexity of reasoning on ADFs, spans the first three levels of the polynomial hierarchy [EFFW07]. This allows for dynamic yet single-shot and complexity-sensitive ASP encodings.

Finally, we motivated and developed the design of the whole pipeline of evaluating collections of strict and defeasible rules expressed in the controlled natural language ACE [FKK08] via the argumentation-inspired direct-stable-semantics defined in [SW17] (Section 4). We, first of all, presented the results of an in-depth investigation of the inner-workings of the main open-source interface to rule systems that exists for ACE, `ACERules` [Kuh07], pinpointing the source of some limitations we found while also highlighting useful features. We then developed an alternative approach that involves simulating general rules also allowing existential quantification via normal rules. We also developed novel dynamic ASP encodings for evaluating (normal) first order defeasible theories via the direct-stable semantics.

We implemented systems that serve as proof-of-concept based on our encodings for ADFs (sections 3.1.4[1] and 3.2.7[2]). We also implemented the rudiments of the pipeline for evaluating strict and defeasible ACE rules via the direct-stable-semantics (Section 4.4)[3]. For our ADF systems we reported on experiments we carried out that complement recent experiments by us that are not part of this work [DKLW18] as well as by other authors [LMN+18a] (see also the discussion in the section on related work and our in-depth survey of all recent empirical evaluations of ADF systems in Section B.3 of Appendix B). For our system for evaluating ACE rules we also presented results on some preliminary experiments that are promising given the complexity of the pipeline we implement.

Related work As mentioned in the introduction to this work, there has been a substantial amount of investigation in implementation strategies and development of systems for abstract argumentation; in particular for Dung's AFs. Thus, the second international competition on computational models of arguments (ICCMA'17)[4] featured 16 solvers for Dung's AFs, with at least 9 solvers participating in each of the 24 tasks resulting from the combination of considered semantics and reasoning problems [GLMW16, AGLMW18]. The third ICCMA will be held this year (2019[5]), the first was held in 2015 [TVC+16, TV17].

Implementation techniques for abstract argumentation can be broadly classified in direct and reduction based, where the direct approach involves the development of native algorithms for the formalism and reasoning problem of interest. The reduction approach is based on the translation of the reasoning problem of interest to some formalism for which systems exist; most notably SAT and QSAT, constraint satisfaction problems, and answer-set programming. A recent survey is [CGTW18]; for an earlier survey more focused on abstract argumentation we refer to [CDG+15].

Reductions to the satisfiability problem for propositional logic (SAT) have been first advocated for Dung's AFs in [DB02] and [DB03] and then further developed in [BD04]. Prominent SAT-based systems that also participated at ICCMA'2017 include `argmat-dvisat`

[1]`https://www.dbai.tuwien.ac.at/proj/adf/qadf/`

[2]`https://www.dbai.tuwien.ac.at/proj/adf/yadf/`

[3]`https://www.dbai.tuwien.ac.at/proj/grappa/emil/`; see also our previous system `dACERules` at `https://www.dbai.tuwien.ac.at/proj/adf/dAceRules/`

[4]See `http://argumentationcompetition.org/index.html`.

[5]`http://www.iccma2019.dmi.unipg.it/index.html`

and `argmat-sat`[6] [PLJ17], jArgSemSAT[7] [CGV14, CVG16, CVG17], and `cegartix`[8] [DJWW14]. QSAT encodings for Dung's AFs have been proposed in [EW06] and [AC13], but these have not materialised in QBF-based systems. Some of the first ASP encodings for Dung's AFs are presented in [NCO08] and [WN08] (see [TS11] for an early and [DDP$^+$18] for a more recent survey), prominent systems include several variants of `ASPARTIX` [EGW10][9] as well as the system `ASPrMin` [FVCG18][10] which participated at ICCMA'17.

For ADFs, as has already been mentioned, the first system was the ASP-based system `ADFSys` [EW12] which lead to the `DIAMOND`[11] family of systems [ES14, ES16]. What all of these systems have in common is that they are based on static encodings and rely on a representation of the acceptance conditions of ADFs as boolean functions. Thus, they need to transform the acceptance conditions in the propositional representation to the functional representation; an operation which may involve an exponential blowup. An alternative approach to `DIAMOND` is using QBF encodings [Dil14] as we have implemented in our system `QADF` [DWW14]. As already mentioned, versions of `QADF` previous to the version 0.4.0 we present in this work do not include encodings for the stable semantics nor allow for making use of information about the link types of ADFs.

A more recent approach for solving reasoning problems for ADFs is that presented in [LMN$^+$18a] and materialised in the system `k++ADF`[12]. This approach, similarly to `cegartix` for AFs, relies on incremental calls to a SAT solver based on the fact that for ADFs which are "close to" being an ADF of a subclass with lower complexity the number of calls to an oracle of the problem of lower complexity can be bounded by a constant. The current version of `k++ADF` in particular is able to detect whether the input ADF is k-bipolar for a sufficiently low value of k. The motivation behind the work in [LMN$^+$18a] is thus similar to that behind the link-information-sensitive QBF encodings we present in Section 3.1.3; the difference being that ours is a full-reduction based method in that we rely also on the QSAT solvers to make use of the information provided by the links while the approach in [LMN$^+$18a] is somewhat of a hybrid in that the calls to the SAT solver are managed directly by the reasoning algorithm. In any case, the experiments reported on in [LMN$^+$18a] suggest that this control over the calls and direct use of a SAT solver allows `k++ADF` to currently outperform all other ADF systems including our own (at least for the admissible and preferred semantics).

We should finally mention that also for the extension of ADFs to arbitrary labelled graphs, GRAPPA [BW14], there has been some work on implementations. In particular, the system reported on in [Ber16] which relies on a translation to ADFs (with a potential

[6]`https://sites.google.com/site/argumatrix/`
[7]`https://sourceforge.net/projects/argsemsat/`
[8] `http://www.dbai.tuwien.ac.at/proj/argumentation/cegartix/`
[9]`https://www.dbai.tuwien.ac.at/research/argumentation/aspartix/`
[10]`https://helios.hud.ac.uk/scommv/storage/ASPrMin-v1.0.tar.gz`
[11]`http://diamond-adf.sourceforge.net/`
[12]`https://www.cs.helsinki.fi/group/coreo/k++adf/`

exponential blowup) and the ASP-based system GrappaVis [HW16][13] which is more focused on the graphical representation of GRAPPA frameworks and their evaluation but also uses static and dynamic ASP-based encodings for evaluating GRAPPA frameworks [Hei16]. The latter, in particular, are based on the same idea as our dynamic ASP encodings for ADFs; we have thus also presented (a somewhat improved version of) these encodings for GRAPPA as part of our work in [BDH$^+$17] and [BDH$^+$].

Turning to work related to that we presented in Section 4, we refer once more to [CGTW18] for an overview of the more heterogeneous landscape of development of implementation strategies for structured argumentation formalisms. See also again [DDP$^+$18] for an overview of realisations of structured argumentation using ASP. The use of ASP in computational linguistics, including for natural language understanding (see e.g. the discussion in [LL13] about the relative merits of first order logic vs ASP for this task) and reasoning on CNLs (as e.g. also in [EÖ15]), is surveyed in [Sch18a].

We have discussed the relationship between our ASP encoding strategy for the direct-stable-semantics and that presented in [SW17][14] at length in Section 4.2.4.2. We have also made detailed reference to alternative reasoners available for ACE, mainly the FOL reasoner RACE [Fuc10] and the system ACERules [Kuh07] in Section 4; we refer to the Attempto Project webpage for further resources and available tools for ACE[15]. We note that the line of study of ACERules has been continued via the PENG$^{\text{ASP}}$ system [GS17][16], which as far as we are able to tell, inherits many of the features (while also improving on several others, e.g. in one of the more recent iterations, using a bi-directional grammar for specifying and verbalising ASP-programs [Sch18b]) of ACERules; in particular, that it does not offer explicit support of existential quantification. Our reasons for initiating our study with the system ACERules were mainly pragmatic: the system PENG$^{\text{ASP}}$ is at the time of writing this work not open-source nor publicly available for experimentation.

Future work The motivation behind our novel QBF and ASP encodings for ADFs was to lay the groundwork for better performing ADF solvers. Current experimental results (see discussions in Section 3.1.4 and Section 3.2.7; also, again, our survey in Section B.3 of Appendix B for the details) suggest some potential for further boosting the performance of our encodings on dense ADFs, but principally that our encodings presently could be of more use as benchmarks driving further development of QSAT and ASP solvers. In any case, more large scale experiments would be useful: considering more benchmarks (ideally, also "real-life" data-sets e.g. ADFs generated from planned user studies with the application reported on in [Neu18]), more solvers (and preprocessors), different parameters in the input, and eventually more semantics as well as reasoning problems.

[13]https://www.dbai.tuwien.ac.at/proj/adf/grappavis/
[14]https://github.com/hstrass/defeasible-rules
[15]http://attempto.ifi.uzh.ch/site/resources/
[16]http://web.science.mq.edu.au/~rolfs/PENG-Light-Answer-Set.html

For our QBF encodings it would in particular also be of value to consider QSAT solvers that do not assume QBFs with the matrix in conjunctive normal form. As detailed in Section 3.1.4 we have already implemented the transformation of some of our encodings generated by our system `QADF` to the QCIR format; it would be useful to hence include solvers that use this input format in future experiments [JKS16]. Depending on results obtained on boosting the performance of `QADF` when using our link-information-sensitive encodings it may make sense to develop encodings similar to our link-information-sensitive encodings but optimised to other subclasses of ADFs, e.g. acyclic and concise ADFs [LMN$^+$18a]. In a similar vein, the strategy behind our dynamic ASP-based encodings could be used to provide encodings for alternative ADF semantics [Pol14] as well as recent generalizations of ADFs (and GRAPPA) such as weighted ADFs [BSWW18] (with finite values).

Regarding our work on realisation of the EMIL pipeline, a pressing issue is to make available the version of our implementation incorporating treatment of the copula "be" and preposition "of". Also, adding support for generating arguments from defeasible theories and experimenting with different ASP grounders and solvers are issues that should be addressed. Other future work is incorporation of means of restricting existential variables whenever possible (in the spirit of `ACERules`) and/or restrictions to ensure finite groundability (see e.g. [AZZ17]). At a more theoretical level, investigating different forms of adding defeasibility (e.g. distinguishing between defeasible rules as normality assumptions and defeasibility introduced by naming conventions as present in the example developed in Section 4.3) as well as negation-as-failure to the formalism of [SW17] (and ACE) are intriguing issues to consider. More long term are investigations of alternative means of supporting existential quantification as well as enhancing the natural language understanding capabilities of our system.

APPENDIX A

Report on QBF'18 workshop study on the effect of different QBF solvers and preprocessors on link information sensitive QBF encodings for ADFs

For the International Workshop on Quantified Boolean Formulas 2018 (QBF'18) we presented the results of an initial empirical study comparing the performance of several important QBF solvers and preprocessors on our link information sensitive QBF encodings presented in Section 3.1.3 and our non link information sensitive encodings from [Dil14, DWW14, DWW15]. We here present the experimental setup and results of this study focusing on credulous reasoning w.r.t. the admissible semantics and skeptical reasoning for the preferred semantics. In Appendix B (in particular Section B.3) we give an overview of all recent empirical evaluations of ADF systems (including those we have contributed to) [BDH$^+$17, Kes17, DKLW18, LMN$^+$18b] also again commenting on how the results in this appendix fit into this larger landscape of empirical evaluations.

A.1 Experimental setup

For the study for QBF'18 we used a modification of the generator for ADFs we implemented for our evaluation from [BDH$^+$17] similar to that of our study comparing performance of several ADF systems on acyclic vs. non acyclic ADFs reported on in [DKLW18] (see also the previous experiments reported on in [Kes17]). We refer to Appendix B for details but also mention the basic functioning of the generator here.

The generator takes an undirected graph as input and returns an ADF inheriting the structure of the graph (edges become links and nodes result in statements of the ADF). Each parent of a statement is assigned to one of 5 different groups (with some probability that can be set by the user of the generator) determining whether the parent participates in a subformula of the statements acceptance condition representing the notions of attack, group-attack, support, or group-support familiar from argumentation (see Appendix B). Also, the parents can appear as literals connected by the connective exclusive-or. Crucial for our study for QBF'18 is that statements appearing in the exclusive-or groups form (together with the statement having the acceptance condition thus formed) dependent links, while statements appearing in the other groups form attacking or supporting links. The type of the links of the generated ADFs can thus be easily extracted from the form of the acceptance conditions.

In our experiments for QBF'18 the input graphs stem from a data-set used at the second international competition on computational models of argumentation (ICCMA), namely from (Dung) AFs generated from traffic networks [Dil17]. More specifically, based on preliminary experiments, we selected 39 AFs at random from a subset of AFs having up to 300 arguments. From the resulting 39 AFs (interpreted as undirected graphs), we generated 39 ADFs with 20% and 39 ADFs with 60% of the parents assigned to the exclusive-or groups (the other parents are assigned to the remaining groups with equal probability). From each of these sets we then generated sets (of 39 ADFs each) with 0%, 25%, 50%, and 75% of the links to be unknown. We generated our link-sensitive encodings for the resulting 312 ADFs both for the admissible and preferred semantics. Finally, we also generated the encodings from [DWW14, DWW15] for the ADFs without information about the links. We thus reach a total of 390 QBF encodings per semantics.

We focused on comparing the performance of several of the QBF solvers top-ranked in the QBF competition QBFEVAL'17[1], together with the preprocessing tools `HQSpre` [WRMB17] as well as `Bloqqer` [HJL$^+$15]: `RAReQS` [JKMC16], `CAQE` [RT15, Ten18], `dynDepQBF` [CW17], and `DepQBF` [LE17]. We also used `DepQBF` with heavy use of preprocessing (`Bloqqer`, `HQSpre`, `QRATPre+` [LE18b]) that was presented at QBFEVAL'18 (`DepQBF`'18). We also considered the outcome of `HQSpre` as well as `Bloqqer` when running alone. Our time-out was 1800 seconds. We refer to Section 3 of [LE18a] for further details on the solvers (and the versions, most of which are special versions for QBFEVAL'17) we used, as the experimental setup is adapted from that used in the study reported on there.

A.2 Results

Tables A.1 to A.5 show the results for different solver + preprocessor combinations for the preferred semantics; tables A.6 to A.10 the results for the admissible semantics. The tables show, for the indicated solver+preprocessor combination and set of QBFs, the percent of instances solved (%S), number of instances solved (#S), number of which are

[1]`http://www.qbflib.org/qbfeval17.php`

satisfiable (#SS) and unsatisfiable (#SU). Also, the total time taken by the solvers (TT), average time (AT), and average time on solved instances (ATS). "O-Z%-X" stands for the "original" (non-link-information-sensitive) encodings from [DWW14], generated based on the ADFs with Z% parents in exclusive-or groups (i.e. dependent links). "N-Z%-X" stands for the same, but with the "new" link-sensitive encodings. "N-Z%-U" stands for the new encodings on the ADFs with Z% unknown link types. The highest percent of instances solved for each data-set in a set of experiments (grouped in one table) is marked in red.

Confirming results in other studies [DWW14, BDH^{+}17, Kes17, DKLW18, LMN^{+}18b] the results for the preferred semantics are rather disappointing with the best performing solver and preprocessor combination (`dynDepQBF` + `Bloqqer`; see Table A.3) solving around 24% of the non-link-information-sensitive encodings and 21% of the link-information-sensitive encodings. In general, the solvers perform somewhat worse on the link-information-sensitive encodings; although the use of pre-processing levels-out the performance in some cases (combination of `dynDepQBF` with `Bloqqer` as well as `dynDepQBF` or `DepQBF` with the preprocessor `HQSpre`; see tables A.3 and A.4). In general, the solvers also seem rather unsensitive to the number of dependent and known vs. unknown links when run on the link-information-sensitive encodings. Something of an exception is `dynDepQBF` when used as a stand-alone solver as it performs better when there are less unknown links (Table A.1).

Also confirming other studies, the results for the admissible semantics are more promising. Thus the best performing solver for this semantics (`DepQBF`'18; i.e. the version of `DepQBF` submitted to QBFEval'18) solves around 99% of the non-link-information-sensitive encodings and around 97% of the link-information-sensitive-encodings (with average solving times under a minute and two minutes respectively; see Table A.10). Interestingly, the preprocessors we considered in the study (in particular `Bloqqer` but also to some degree `HQSpre`) seem to latch on to the information provided by the links (Table A.7). Thus, for instance, `Bloqqer` used as a stand-alone-tool is able to solve e.g. around 77% of the link-information-sensitive encodings (in under 3 seconds on average) for ADFs with 20% dependent links vs. 67% of the non-link-information-sensitive encodings. Although for the best performing tool (`DepQBF`'18) this is not the case, for several of the solvers the use of preprocessing (especially with `HQSpre` as can be seen in Table A.7 but also particularly considering the mean running time on solved instances when using `Bloqqer`; see Table A.8) then improves their performance on the link-information-sensitive vs. non-link-information sensitive encodings.

To conclude, at least on the ADFs we considered in our study for QBF'18, reasoning for the preferred semantics does not seem to be eased by information about the structure of the ADFs. On the other hand, for the admissible semantics, such information does improve the performance of some systems (when used with preprocessors). This is consistent with studies on the performance of `QADF` (among other systems) on acyclic (ADFs having an underlying acyclic graph) vs. non acyclic ADFs carried out in [DKLW18], where there are gains of performance on the acyclic instances for the admissible semantics but no such gains in performance can be observed for the preferred semantics. On the other hand, the impressive gains obtained by the best performing system on the admissible semantics in the study which we refer to here (`DepQBF`'18), do not seem to arise from the information about the links. Such gains are in fact also a novelty w.r.t. other studies on ADF systems which have used (the non-link-information-sensitive encodings produced by) `QADF` in combination with alternative versions of `DepQBF` and `Bloqqer` (`Bloqqer` 035 and `DepQBF` 4.0 in [DKLW18], while `Bloqqer` 037 and `DepQBF` 6.03 were used in [LMN$^+$18b]). E.g. `QADF` in the study of [DKLW18] had 25 of 100 time-outs (600 seconds) on a (larger non-acyclic) set of ADFs resulting from transportation networks for the admissible semantics; in the study in [LMN$^+$18b] there are 37 of 100 time-outs (1800 seconds). We refer again to Section B.3 in Appendix B for a survey of recent results comparing performance of ADF systems and a brief discussion also of the results detailed in this appendix in light thereof.

		%S	#S	#SS	#SU	TT	MT	AT	ATS
	O-20%-X	0.00	0/39	0	0	70200.00	NaN	1800.00	NaN
	O-60%-X	0.00	0/39	0	0	70200.00	NaN	1800.00	NaN
	N-20%-X	0.00	0/156	0	0	280800.00	NaN	1800.00	NaN
`CAQE`	N-60%-X	0.00	0/156	0	0	280800.00	NaN	1800.00	NaN
	N-0%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-25%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-50%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-75%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN

		%S	#S	#SS	#SU	TT	MT	AT	ATS
	O-20%-X	10.26	4/39	2	2	64538.43	320.16	1654.83	384.61
	O-60%-X	5.13	2/39	2	0	66605.60	2.80	1707.84	2.80
	N-20%-X	5.13	8/156	0	8	266404.58	0.30	1707.72	0.57
`DepQBF`	N-60%-X	5.13	8/156	8	0	266410.07	1.24	1707.76	1.26
	N-0%-U	5.13	4/78	2	2	133203.29	0.90	1707.73	0.82
	N-25%-U	5.13	4/78	2	2	133202.49	0.50	1707.72	0.62
	N-50%-U	5.13	4/78	2	2	133204.18	0.89	1707.75	1.05
	N-75%-U	5.13	4/78	2	2	133204.69	1.40	1707.75	1.17

		%S	#S	#SS	#SU	TT	MT	AT	ATS
	O-20%-X	**20.51**	8/39	4	4	59011.06	70.20	1513.10	401.38
	O-60%-X	**20.51**	8/39	7	1	56269.45	43.55	1442.81	58.68
	N-20%-X	**11.54**	18/156	4	14	252260.99	62.32	1617.06	214.50
`dynDepQBF`	N-60%-X	**11.54**	18/156	16	2	253361.14	43.22	1624.11	275.62
	N-0%-U	**17.95**	14/78	9	5	119479.67	62.32	1531.79	305.69
	N-25%-U	**12.82**	10/78	6	4	124926.34	50.85	1601.62	252.63
	N-50%-U	**7.69**	6/78	2	4	130181.22	11.76	1668.99	96.87
	N-75%-U	**7.69**	6/78	3	3	131034.90	72.84	1679.93	239.15

		%S	#S	#SS	#SU	TT	MT	AT	ATS
	O-20%-X	0.00	0/39	0	0	70200.00	NaN	1800.00	NaN
	O-60%-X	0.00	0/39	0	0	70200.00	NaN	1800.00	NaN
	N-20%-X	0.00	0/156	0	0	280800.00	NaN	1800.00	NaN
`RAReQS`	N-60%-X	0.00	0/156	0	0	280800.00	NaN	1800.00	NaN
	N-0%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-25%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-50%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-75%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN

Table A.1: Results for the preferred semantics. Stand-alone solvers from QBFEVAL'17.

		%S	#S	#SS	#SU	TT	MT	AT	ATS
Bloqqer	O-20%-X	5.13	2/39	0	2	66600.28	0.14	1707.70	0.14
	O-60%-X	2.56	1/39	1	0	68400.09	0.09	1753.85	0.09
	N-20%-X	0.00	0/156	0	0	280800.00	NaN	1800.00	NaN
	N-60%-X	0.00	0/156	0	0	280800.00	NaN	1800.00	NaN
	N-0%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-25%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-50%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-75%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN

		%S	#S	#SS	#SU	TT	MT	AT	ATS
HQSpre	O-20%-X	0.00	0/39	0	0	70200.00	NaN	1800.00	NaN
	O-60%-X	0.00	0/39	0	0	70200.00	NaN	1800.00	NaN
	N-20%-X	0.00	0/156	0	0	280800.00	NaN	1800.00	NaN
	N-60%-X	0.00	0/156	0	0	280800.00	NaN	1800.00	NaN
	N-0%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-25%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-50%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-75%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN

Table A.2: Results for the preferred semantics. Stand-alone preprocessors.

		%S	#S	#SS	#SU	TT	MT	AT	ATS
CAQE	O-20%-X	20.51	8/39	5	3	56056.77	1.93	1437.35	32.10
	O-60%-X	20.51	8/39	8	0	56118.79	2.44	1438.94	39.85
	N-20%-X	5.13	8/156	0	8	266431.93	4.29	1707.90	3.99
	N-60%-X	5.13	8/156	8	0	266407.48	0.71	1707.74	0.94
	N-0%-U	5.13	4/78	2	2	133210.37	2.85	1707.83	2.59
	N-25%-U	5.13	4/78	2	2	133209.50	2.09	1707.81	2.38
	N-50%-U	5.13	4/78	2	2	133208.85	1.74	1707.81	2.21
	N-75%-U	5.13	4/78	2	2	133210.69	2.72	1707.83	2.67

		%S	#S	#SS	#SU	TT	MT	AT	ATS
DepQBF	O-20%-X	20.51	8/39	4	4	55816.24	0.39	1431.19	2.03
	O-60%-X	23.08	9/39	8	1	54050.36	0.39	1385.91	5.60
	N-20%-X	14.74	23/156	11	12	242258.47	49.09	1552.94	124.28
	N-60%-X	17.31	27/156	27	0	237515.34	23.39	1522.53	196.86
	N-0%-U	14.10	11/78	8	3	121477.69	37.49	1557.41	79.79
	N-25%-U	16.67	13/78	10	3	119880.92	34.29	1536.93	221.61
	N-50%-U	16.67	13/78	10	3	119682.01	23.39	1534.38	206.31
	N-75%-U	16.67	13/78	10	3	118733.19	64.20	1522.22	133.32

		%S	#S	#SS	#SU	TT	MT	AT	ATS
dynDepQBF	O-20%-X	25.64	10/39	6	4	52798.03	5.32	1353.80	59.80
	O-60%-X	23.08	9/39	8	1	54249.00	3.21	1391.00	27.67
	N-20%-X	20.51	32/156	16	16	224067.58	16.08	1436.33	27.11
	N-60%-X	20.51	32/156	28	4	223740.53	12.89	1434.23	16.89
	N-0%-U	20.51	16/78	11	5	111912.43	11.21	1434.77	19.53
	N-25%-U	20.51	16/78	11	5	111926.02	12.95	1434.95	20.38
	N-50%-U	20.51	16/78	11	5	111908.16	18.51	1434.72	19.26
	N-75%-U	20.51	16/78	11	5	112061.50	15.02	1436.69	28.84

		%S	#S	#SS	#SU	TT	MT	AT	ATS
RAReQS	O-20%-X	17.95	7/39	3	4	58551.78	9.30	1501.33	135.97
	O-60%-X	15.38	6/39	6	0	59635.75	2.44	1529.12	39.29
	N-20%-X	5.77	9/156	1	8	264934.74	0.69	1698.30	37.19
	N-60%-X	5.77	9/156	9	0	265977.39	0.09	1704.98	153.04
	N-0%-U	5.13	4/78	2	2	133201.40	0.35	1707.71	0.35
	N-25%-U	6.41	5/78	3	2	132777.98	0.59	1702.28	275.60
	N-50%-U	5.13	4/78	2	2	133201.28	0.25	1707.71	0.32
	N-75%-U	6.41	5/78	3	2	131731.47	0.69	1688.87	66.29

Table A.3: Results for the preferred semantics. Solvers from QBFEVAL'17 in combination with `Bloqqer`.

		%S	#S	#SS	#SU	TT	MT	AT	ATS
CAQE	O-20%-X	15.38	6/39	4	2	60863.97	31.44	1560.61	243.99
	O-60%-X	17.95	7/39	7	0	58522.20	1.44	1500.57	131.74
	N-20%-X	5.13	8/156	0	8	266475.71	9.21	1708.18	9.46
	N-60%-X	6.41	10/156	10	0	264349.10	1.11	1694.55	154.91
	N-0%-U	6.41	5/78	3	2	132960.11	6.74	1704.62	312.02
	N-25%-U	6.41	5/78	3	2	131423.10	3.27	1684.91	4.62
	N-50%-U	5.13	4/78	2	2	133220.22	4.07	1707.95	5.05
	N-75%-U	5.13	4/78	2	2	133221.38	5.51	1707.97	5.35

		%S	#S	#SS	#SU	TT	MT	AT	ATS
DepQBF	O-20%-X	20.51	8/39	4	4	56542.18	8.53	1449.80	92.77
	O-60%-X	20.51	8/39	7	1	56573.91	5.79	1450.61	96.74
	N-20%-X	18.59	29/156	13	16	234514.19	51.81	1503.30	203.94
	N-60%-X	19.23	30/156	28	2	232453.79	5.14	1490.09	188.46
	N-0%-U	19.23	15/78	10	5	116617.41	29.20	1495.10	214.49
	N-25%-U	17.95	14/78	10	4	118202.09	16.08	1515.41	214.44
	N-50%-U	19.23	15/78	11	4	115140.95	10.78	1476.17	116.06
	N-75%-U	19.23	15/78	10	5	117007.52	52.19	1500.10	240.50

		%S	#S	#SS	#SU	TT	MT	AT	ATS
dynDepQBF	O-20%-X	20.51	8/39	4	4	56207.99	8.56	1441.23	51.00
	O-60%-X	20.51	8/39	7	1	55898.40	9.71	1433.29	12.30
	N-20%-X	19.23	30/156	16	14	229755.60	8.01	1472.79	98.52
	N-60%-X	20.51	32/156	28	4	229983.55	38.12	1474.25	211.99
	N-0%-U	20.51	16/78	11	5	113281.05	36.73	1452.32	105.07
	N-25%-U	20.51	16/78	11	5	114036.23	26.68	1462.00	152.26
	N-50%-U	19.23	15/78	11	4	116314.51	29.91	1491.21	194.30
	N-75%-U	19.23	15/78	11	4	116107.36	19.44	1488.56	180.49

		%S	#S	#SS	#SU	TT	MT	AT	ATS
RAReQS	O-20%-X	10.26	4/39	2	2	63894.42	3.40	1638.32	223.61
	O-60%-X	10.26	4/39	4	0	63000.83	0.12	1615.41	0.21
	N-20%-X	5.13	8/156	0	8	266445.42	5.66	1707.98	5.68
	N-60%-X	5.13	8/156	8	0	266424.51	2.53	1707.85	3.06
	N-0%-U	5.13	4/78	2	2	133217.19	4.48	1707.91	4.30
	N-25%-U	5.13	4/78	2	2	133214.48	4.02	1707.88	3.62
	N-50%-U	5.13	4/78	2	2	133219.46	5.88	1707.94	4.87
	N-75%-U	5.13	4/78	2	2	133218.80	5.66	1707.93	4.70

Table A.4: Results for the preferred semantics. Solvers from QBFEVAL'17 in combination with HQSpre.

		%S	#S	#SS	#SU	TT	MT	AT	ATS
DepQBF'18	O-20%-X	17.95	7/39	4	3	58880.61	39.51	1509.76	182.94
	O-60%-X	23.08	9/39	8	1	55141.53	24.89	1413.89	126.84
	N-20%-X	9.62	15/156	5	10	260077.86	12.13	1667.17	418.52
	N-60%-X	15.38	24/156	24	0	247169.29	313.87	1584.42	398.72
	N-0%-U	12.82	10/78	7	3	126868.14	308.60	1626.51	446.81
	N-25%-U	11.54	9/78	7	2	126493.60	205.01	1621.71	254.84
	N-50%-U	12.82	10/78	8	2	125651.11	284.02	1610.91	325.11
	N-75%-U	12.82	10/78	7	3	128234.29	309.51	1644.03	583.43

		%S	#S	#SS	#SU	TT	MT	AT	ATS
DepQBF'18 + Bloqqer	O-20%-X	20.51	8/39	4	4	56137.79	61.26	1439.43	42.22
	O-60%-X	23.08	9/39	8	1	54288.62	27.54	1392.02	32.07
	N-20%-X	14.74	23/156	12	11	244774.24	82.25	1569.07	233.66
	N-60%-X	16.67	26/156	26	0	241702.27	68.06	1549.37	296.24
	N-0%-U	12.82	10/78	8	2	123945.38	27.44	1589.04	154.54
	N-25%-U	16.67	13/78	10	3	120293.65	68.73	1542.23	253.36
	N-50%-U	16.67	13/78	10	3	120801.93	67.38	1548.74	292.46
	N-75%-U	16.67	13/78	10	3	121435.54	175.74	1556.87	341.20

Table A.5: Results for the preferred semantics. DepQBF'18 alone and in combination with Bloqqer.

		%S	#S	#SS	#SU	TT	MT	AT	ATS
CAQE	O-20%-X	5.13	2/39	0	2	66721.34	60.67	1710.80	60.67
	O-60%-X	5.13	2/39	2	0	67908.13	654.07	1741.23	654.07
	N-20%-X	0.00	0/156	0	0	280800.00	NaN	1800.00	NaN
	N-60%-X	0.00	0/156	0	0	280800.00	NaN	1800.00	NaN
	N-0%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-25%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-50%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-75%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN

		%S	#S	#SS	#SU	TT	MT	AT	ATS
DepQBF	O-20%-X	23.08	9/39	4	5	54596.86	4.59	1399.92	66.32
	O-60%-X	20.51	8/39	6	2	56369.05	36.45	1445.36	71.13
	N-20%-X	5.13	8/156	0	8	266419.19	1.70	1707.82	2.40
	N-60%-X	5.13	8/156	8	0	266700.42	12.87	1709.62	37.55
	N-0%-U	5.13	4/78	2	2	133215.97	3.29	1707.90	3.99
	N-25%-U	5.13	4/78	2	2	133220.71	4.75	1707.96	5.18
	N-50%-U	5.13	4/78	2	2	133267.12	9.42	1708.55	16.78
	N-75%-U	5.13	4/78	2	2	133415.81	21.50	1710.46	53.95

		%S	#S	#SS	#SU	TT	MT	AT	ATS
dynDepQBF	O-20%-X	41.03	16/39	8	8	47560.92	2.61	1219.51	385.06
	O-60%-X	35.90	14/39	9	5	46507.24	0.76	1192.49	107.66
	N-20%-X	26.28	41/156	21	20	211749.27	6.46	1357.37	115.84
	N-60%-X	23.72	37/156	26	11	219030.29	6.19	1404.04	130.55
	N-0%-U	26.92	21/78	12	9	106006.43	4.86	1359.06	162.21
	N-25%-U	26.92	21/78	13	8	105082.79	7.96	1347.22	118.23
	N-50%-U	24.36	19/78	11	8	108112.60	6.26	1386.06	100.66
	N-75%-U	21.79	17/78	11	6	111577.74	10.21	1430.48	104.57

		%S	#S	#SS	#SU	TT	MT	AT	ATS
RAReQS	O-20%-X	5.13	2/39	0	2	66602.90	1.45	1707.77	1.45
	O-60%-X	5.13	2/39	2	0	66744.19	72.10	1711.39	72.10
	N-20%-X	0.00	0/156	0	0	280800.00	NaN	1800.00	NaN
	N-60%-X	0.00	0/156	0	0	280800.00	NaN	1800.00	NaN
	N-0%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-25%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-50%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN
	N-75%-U	0.00	0/78	0	0	140400.00	NaN	1800.00	NaN

Table A.6: Results for the admissible semantics. Stand-alone solvers from QBFEVAL'17.

		%S	#S	#SS	#SU	TT	MT	AT	ATS
Bloqqer	O-20%-X	66.67	26/39	17	9	23422.40	0.20	600.57	0.86
	O-60%-X	66.67	26/39	20	6	23416.29	0.15	600.42	0.63
	N-20%-X	76.92	120/156	60	60	65142.43	1.30	417.58	2.85
	N-60%-X	67.95	106/156	81	25	90271.31	1.25	578.66	2.56
	N-0%-U	75.64	59/78	36	23	34336.02	1.29	440.21	2.31
	N-25%-U	71.79	56/78	35	21	39727.06	1.25	509.32	2.27
	N-50%-U	71.79	56/78	35	21	39750.16	1.10	509.62	2.68
	N-75%-U	70.51	55/78	35	20	41600.50	1.69	533.34	3.65

		%S	#S	#SS	#SU	TT	MT	AT	ATS
HQSpre	O-20%-X	2.56	1/39	0	1	68400.00	0.00	1753.85	0.00
	O-60%-X	2.56	1/39	1	0	68400.00	0.00	1753.85	0.00
	N-20%-X	16.67	26/156	22	4	234315.57	4.74	1502.02	12.14
	N-60%-X	4.49	7/156	7	0	268242.67	0.09	1719.50	6.10
	N-0%-U	14.10	11/78	10	1	120730.22	4.29	1547.82	11.84
	N-25%-U	8.97	7/78	6	1	127847.05	0.79	1639.06	6.72
	N-50%-U	8.97	7/78	6	1	127854.34	0.99	1639.16	7.76
	N-75%-U	10.26	8/78	7	1	126126.63	3.55	1617.01	15.83

Table A.7: Results for the admissible semantics. Stand-alone preprocessors.

		%S	#S	#SS	#SU	TT	MT	AT	ATS
	O-20%-X	79.49	31/39	17	14	14465.18	0.00	370.90	2.10
	O-60%-X	79.49	31/39	21	10	16294.67	0.00	417.81	61.12
	N-20%-X	80.77	126/156	60	66	54023.03	0.00	346.30	0.18
CAQE	N-60%-X	73.72	115/156	84	31	73847.32	0.00	473.38	0.41
	N-0%-U	79.49	62/78	37	25	28801.76	0.00	369.25	0.03
	N-25%-U	**78.21**	61/78	37	24	30632.12	0.00	392.72	0.53
	N-50%-U	76.92	60/78	35	25	32436.07	0.00	415.85	0.60
	N-75%-U	74.36	58/78	35	23	36000.40	0.00	461.54	0.01

		%S	#S	#SS	#SU	TT	MT	AT	ATS
	O-20%-X	79.49	31/39	17	14	14408.37	0.00	369.45	0.27
	O-60%-X	69.23	27/39	21	6	23279.94	0.00	596.92	62.22
	N-20%-X	80.13	125/156	60	65	55848.97	0.00	358.01	0.39
DepQBF	N-60%-X	71.79	112/156	82	30	79956.33	0.00	512.54	6.75
	N-0%-U	78.21	61/78	36	25	30604.69	0.00	392.37	0.08
	N-25%-U	76.92	60/78	36	24	33188.91	0.00	425.50	13.15
	N-50%-U	74.36	58/78	35	23	36000.10	0.00	461.54	0.00
	N-75%-U	74.36	58/78	35	23	36011.60	0.00	461.69	0.20

		%S	#S	#SS	#SU	TT	MT	AT	ATS
	O-20%-X	74.36	29/39	17	12	18195.86	0.00	466.56	6.75
	O-60%-X	71.79	28/39	22	6	20042.72	0.00	513.92	8.67
	N-20%-X	78.21	122/156	60	62	61475.70	0.00	394.08	2.26
dynDepQBF	N-60%-X	73.08	114/156	83	31	76016.10	0.01	487.28	3.65
	N-0%-U	76.92	60/78	36	24	32419.10	0.01	415.63	0.32
	N-25%-U	75.64	59/78	36	23	34248.78	0.00	439.09	0.83
	N-50%-U	76.92	60/78	36	24	32971.24	0.00	422.71	9.52
	N-75%-U	73.08	57/78	35	22	37852.69	0.01	485.29	0.92

		%S	#S	#SS	#SU	TT	MT	AT	ATS
	O-20%-X	**89.74**	35/39	17	18	7200.00	0.00	184.62	0.00
	O-60%-X	**82.05**	32/39	21	11	12642.59	0.00	324.17	1.33
	N-20%-X	**82.05**	128/156	60	68	50400.10	0.00	323.08	0.00
RAReQS	N-60%-X	**76.28**	119/156	84	35	66624.80	0.00	427.08	0.21
	N-0%-U	**80.77**	63/78	37	26	27004.20	0.00	346.21	0.07
	N-25%-U	**78.21**	61/78	37	24	30615.60	0.00	392.51	0.26
	N-50%-U	**79.49**	62/78	35	27	28802.70	0.00	369.27	0.04
	N-75%-U	**78.21**	61/78	35	26	30602.40	0.00	392.34	0.04

Table A.8: Results for the admissible semantics. Solvers from QBFEVAL'17 in combination with `Bloqqer`.

		%S	#S	#SS	#SU	TT	MT	AT	ATS
	O-20%-X	**53.85**	21/39	3	18	32937.62	4.87	844.55	25.60
	O-60%-X	**46.15**	18/39	6	12	41391.19	17.14	1061.31	199.51
	N-20%-X	**58.33**	91/156	25	66	135483.32	13.59	868.48	203.11
CAQE	N-60%-X	44.87	70/156	40	30	160209.61	27.53	1026.98	77.28
	N-0%-U	**55.13**	43/78	18	25	68883.59	13.59	883.12	136.83
	N-25%-U	**56.41**	44/78	18	26	67624.37	17.02	866.98	146.01
	N-50%-U	50.00	39/78	16	23	75892.68	30.36	972.98	145.97
	N-75%-U	44.87	35/78	13	22	83292.30	38.01	1067.85	168.35

		%S	#S	#SS	#SU	TT	MT	AT	ATS
	O-20%-X	20.51	8/39	4	4	55973.21	4.53	1435.21	21.65
	O-60%-X	20.51	8/39	6	2	55846.89	2.07	1431.97	5.86
	N-20%-X	31.41	49/156	29	20	196124.77	0.86	1257.21	71.93
DepQBF	N-60%-X	27.56	43/156	35	8	204199.88	7.36	1308.97	18.60
	N-0%-U	33.33	26/78	19	7	95120.25	4.20	1219.49	58.47
	N-25%-U	33.33	26/78	19	7	94464.17	6.79	1211.08	33.24
	N-50%-U	26.92	21/78	14	7	103990.99	4.68	1333.22	66.24
	N-75%-U	24.36	19/78	12	7	106749.24	2.58	1368.58	28.91

		%S	#S	#SS	#SU	TT	MT	AT	ATS
	O-20%-X	38.46	15/39	8	7	43681.18	1.26	1120.03	32.08
	O-60%-X	33.33	13/39	8	5	48488.78	1.17	1243.30	129.91
	N-20%-X	56.41	88/156	46	42	128790.40	3.56	825.58	72.62
dynDepQBF	N-60%-X	**51.28**	80/156	56	24	141684.38	4.29	908.23	61.05
	N-0%-U	53.85	42/78	26	16	66520.32	2.24	852.82	40.96
	N-25%-U	**56.41**	44/78	27	17	63214.56	7.28	810.44	45.79
	N-50%-U	**56.41**	44/78	27	17	66625.21	6.33	854.17	123.30
	N-75%-U	**48.72**	38/78	22	16	74114.69	2.11	950.19	55.65

		%S	#S	#SS	#SU	TT	MT	AT	ATS
	O-20%-X	51.28	20/39	4	16	36520.31	1.55	936.42	116.02
	O-60%-X	35.90	14/39	6	8	45808.24	6.70	1174.57	57.73
	N-20%-X	49.36	77/156	25	52	151597.11	8.48	971.78	122.04
RAReQS	N-60%-X	42.31	66/156	39	27	166454.79	11.01	1067.02	67.50
	N-0%-U	50.00	39/78	18	21	73347.30	9.20	940.35	80.70
	N-25%-U	47.44	37/78	18	19	76586.13	9.66	981.87	75.30
	N-50%-U	43.59	34/78	16	18	83612.33	10.33	1071.95	129.77
	N-75%-U	42.31	33/78	12	21	84506.15	8.93	1083.41	106.25

Table A.9: Results for the admissible semantics. Solvers from QBFEVAL'17 in combination with `HQSpre`.

		%S	#S	#SS	#SU	TT	MT	AT	ATS
DepQBF'18	O-20%-X	**100.00**	39/39	19	20	1487.59	3.89	38.14	38.14
	O-60%-X	**97.44**	38/39	26	12	3597.13	8.97	92.23	47.29
	N-20%-X	**98.08**	153/156	74	79	18481.91	14.32	118.47	85.50
	N-60%-X	**96.79**	151/156	104	47	25083.97	12.57	160.79	106.52
	N-0%-U	**98.72**	77/78	45	32	8365.77	10.62	107.25	85.27
	N-25%-U	**94.87**	74/78	43	31	14651.92	12.62	187.85	100.70
	N-50%-U	**97.44**	76/78	45	31	10952.71	13.84	140.42	96.75
	N-75%-U	**98.72**	77/78	45	32	9595.47	13.82	123.02	101.24

		%S	#S	#SS	#SU	TT	MT	AT	ATS
DepQBF'18 + Bloqqer	O-20%-X	97.44	38/39	19	19	2690.10	0.06	68.98	23.42
	O-60%-X	94.87	37/39	26	11	4842.12	0.06	124.16	33.57
	N-20%-X	91.67	143/156	69	74	27051.03	0.06	173.40	25.53
	N-60%-X	92.95	145/156	100	45	24754.97	0.06	158.69	34.17
	N-0%-U	93.59	73/78	42	31	10465.57	0.05	134.17	20.08
	N-25%-U	89.74	70/78	41	29	16196.03	0.06	207.64	25.66
	N-50%-U	92.31	72/78	43	29	12963.08	0.06	166.19	30.04
	N-75%-U	93.59	73/78	43	30	12181.31	0.06	156.17	43.58

Table A.10: Results for the admissible semantics. DepQBF'18 alone and in combination with Bloqqer.

APPENDIX B

Report on AAAI'17 study comparing performance of main ADF solvers with our dynamic ASP encoding based system

For our work presented at the AAAI'17 conference [BDH$^+$17] we carried out experiments to compare the performance of our implementation of the dynamic ASP based solving strategy for ADFs presented in Section 3.2 with the then existing systems for ADFs. Specifically, we compared the performance of our prototype system, `YADF` (Section 3.2.7), with that of the static ASP based system `DIAMOND` [ES14] and our QBF based system `QADF` [DWW14]. We focused on credulous and skeptical reasoning for the admissible and preferred semantics, respectively. We here, first of all, present the experimental setup and results of the work we reported on at AAAI'17. We then survey subsequent experimental evaluations of ADF systems with which we have collaborated as well as by colleagues and discuss the results from AAAI'17 in light of these.

B.1 Experimental setup

To generate ADFs, in our study for AAAI'17 we first used a "grid-based" ADF generator which has been employed in the then only evaluations of ADF systems known to us [Ell12, Dil14, DWW14]. The basic idea behind this generator is that a predetermined number of statements are placed on a grid with directed edges and a certain width, an outgoing edge from one neighbor to the other indicating that the one neighbor has a certain probability to appear in the acceptance condition of the other. Apart from this aspect, there are probabilities assigned to the links in the grid governing whether both statements in a link appear in the acceptance condition of each other ("symmetric relation of attack or support"), whether the statement is mutated into one of the truth constants $\perp$ or $\top$, and whether each of the statements (or constant in case a statement has been mutated) in question will appear negated or not in the relevant acceptance condition. Finally, for constructing the acceptance conditions there is a probability that determines whether the connective appearing between the part of the acceptance condition that has already been constructed and a new statement that is added to the condition is a conjunction ($\wedge$) or disjunction ($\vee$).

Regarding the parameters governing the ADF generation process of the grid based generator, for our experiments we used the default values of the generator: 7 for the width of the grid[1], 0.5 for the probability of symmetry in relation of attack or support, 0.2 for the probability of a variable being changed into a constant, and 0.5 for the probability of a given connective being a conjunction or disjunction when constructing the acceptance conditions. We produced 40 ADFs with 10, 20, 30, 40, 50, 60, 70, and up to 80 statements with the grid based generator; i.e. 320 ADFs in total.

The grid-based generator has the drawback that the underlying grid structure is somewhat arbitrary and the control over the form of the acceptance conditions generated rather limited. For these reasons, we also wrote our own graph-based generator[2]. This generator takes any desired directed graph as input and generates an ADF inheriting the structure of the graph. This means that the edges of the graph become links and the nodes become statements of the resulting ADF.

For constructing the acceptance conditions of the ADFs, the graph based generator assigns each of the parents of a statement to one of 5 different groups (with equal probability in our experiments). This assignment determines whether the parent participates in a subformula of the statement's acceptance condition representing the notions of attack, group-attack, support, or group-support familiar from argumentation. Also, the parents can appear as literals connected by the exclusive-or connective ($\oplus$; this, in order to capture the full complexity of ADFs). More precisely, if for a statement s_0, the parents $s_1, \ldots, s_n$ are assigned to the group for attack, the corresponding subformula for these parents in the acceptance condition of s_0 has the form:

$$\neg s_1 \wedge \ldots \wedge \neg s_n$$

The subformulas for group-attack, support, group-support, and the exclusive-or group on the other hand have the form

$$\neg s_1 \vee \ldots \vee \neg s_n,$$

[1]This means a statement can have up to eight parents.

[2]This generator underlies the subsequent version available at `https://www.dbai.tuwien.ac.at/proj/grappa/subadfgen/`.

	Cred-*adm*			Skept-*prf*		
	DIAMOND	QADF	YADF	DIAMOND	QADF	YADF
Gri-10	0.11 (0)	0.62 (0)	0.66 (0)	0.31 (0)	0.9 (0)	0.75 (0)
Gri-20	0.35 (0)	0.8 (0)	0.96 (0)	51.17 (20)	41.53 (0)	1.26 (0)
Gri-30	0.9 (0)	1.01 (0)	1.13 (0)	51.48 (38)	497.4 (39)	1.76 (0)
Gri-40	1.64 (0)	1.21 (0)	1.34 (0)	- (40)	- (40)	2.68 (0)
Gri-50	2.8 (0)	1.47 (0)	1.52 (0)	- (40)	- (40)	4.83 (0)
Gri-60	4.3 (0)	2.08 (0)	1.86 (0)	- (40)	- (40)	9.6 (0)
Gri-70	6.52 (0)	3.52 (0)	2.08 (0)	- (40)	- (40)	68.48 (1)
Gri-80	8.83 (0)	3.08 (1)	2.37 (0)	- (40)	- (40)	84.37 (6)
Metro	5.7 (0)	5.86 (7)	1.6 (0)	- (40)	- (40)	43.01 (11)

Table B.1: Mean running times in seconds for credulous reasoning under the admissible (Cred-*adm*) and skeptical reasoning under the preferred (Skept-*prf*) semantics. On ADF instances generated by the grid-based (Gri-X = ADFs with X statements) and graph-based generator (5 ADFs per city). Number of time-outs (out of 40 instances; with time-out of 600 seconds) in parentheses. Mean running times are computed disregarding time-outs.

$$s_1 \vee \ldots \vee s_n,$$
$$s_1 \wedge \ldots \wedge s_n,$$

and

$$l_1 \oplus \ldots \oplus l_n$$

respectively. In the last suformula, l_i ($1 \leq i \leq n$) is either s_i or $\neg s_i$ with equal probability. Also, for groups to which no parents are assigned, the corresponding subformulas are $\top$ or $\bot$ with identical probability. To generate the final acceptance condition of a statement, the subformulas for the different groups of parents of the statement are connected via $\wedge$ or $\vee$; again, with equal probability.

In our experiments the input graphs we used for our graph based generator represent public transport networks of 8 different cities: metro networks of Berlin, Beijing, London, Munich, Shanghai, Singapore, and Vienna, as well as the tram network of Vienna. We generated 5 ADFs per city. The motivation behind using metro networks is that these may, to some extent, resemble the structure of more or less complex "real life" discussions, where usually there are many claims but not as many relations between the claims. I.e. discussions usually progress by considering a few claims at a time; then, newly expounded claims are considered.

Experiments were carried out on a 48 GB `Debian` (8.5) machine with 8 `Intel Xeon` processors (2.33 GHz). Due to known problems with the available versions of `DIAMOND`

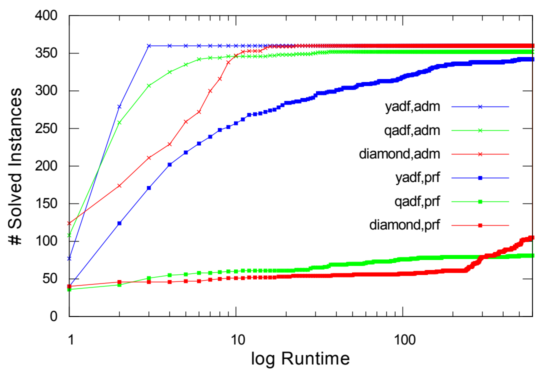

Figure B.1: Number of instances solved in running time less than x seconds for credulous reasoning under the admissible and skeptical reasoning under the preferred semantics. All grid-based and graph-based instances were considered (360 total).

at the time[3], 2.0.2 and 2.0.0, we used version 0.9 (modified to support credulous and skeptical reasoning). For QADF we used version 0.3.2 with the preprocessing tool bloqqer 035 [HJL$^+$15][4] and the QSAT solver DepQBF 4.0 [LB10, LE17][5]. This version of QADF includes the (non-QDIMACS) intermediate representation of the QBF encodings as is also included in the version described in Section 3.1.4 (version 0.4.0), but does not produce the link-information-sensitive encodings. YADF is version 0.1.0 with the rule decomposition tool lpopt version 2.0 [BMW16a][6] and the ASP solver clingo 4.4.0 [GKK$^+$18][7]. Version 0.1.0 of YADF is identical to the version (0.1.1) presented in Section 3.2.7, except that it does not generate the encodings for the stable semantics. We set the time-out for our experiments to be of 600 seconds.

B.2 Results

Table B.1 and Figure B.1 summarise the results of our study for AAAI'17. As can be seen from these, YADF (0.1.0) performed comparably, or even somewhat better, to DIAMOND (0.9) and QADF (0.3.2; non-link-information-sensitive encodings) for credulous reasoning under the admissible semantics. There is a clear advantage in the use of the dynamic ASP-based encodings over DIAMOND and QADF for skeptical reasoning under the preferred semantics, although there are 7 time-outs on the grid based instances with 70 and 80 statements as well as on 11 of the metro-based instances.

[3]See discussion in [SE17]. In particular, the version of DIAMOND reported there (goDIAMOND) was not yet available at the time of submission of our work to AAAI'17.

[4]http://fmv.jku.at/bloqqer/

[5]http://lonsing.github.io/depqbf/

[6]https://www.dbai.tuwien.ac.at/research/project/lpopt/

[7]https://potassco.org/clingo/

B.3 Survey of subsequent experimental evaluations of ADF systems and assessment of AAAI'17 results in light thereof

Since our experimental evaluation for AAAI'17, there have been three noteworthy experimental evaluations of ADF systems including our system YADF. All of these evaluations build on the experimental setup of our evaluation for AAAI'17; in particular they also consider credulous reasoning for the admissible semantics as well as skeptical reasoning for the preferred semantics. Yet, the experiments also include the most recent version of DIAMOND as well as a novel system for ADFs based on incremental SAT solving; we have collaborated with two of these evaluations [Kes17, DKLW18], while the other has been carried out by colleagues of ours [LMN$^+$18a]. We briefly summarise the setup and results of the mentioned evaluations to then attempt to paint a general picture of what is currently known about the performance of ADF systems, which includes those we present in this work.

In [Kes17] the author compares the performance of ADF systems on acyclic, being those whose underlying graph is acyclic, vs. non-acyclic ADFs (i.e. ADFs whose underlying graph is not guaranteed to be acyclic). As is proven in [Kes17, DKLW18], for acyclic ADFs, the grounded interpretations, complete interpretations, preferred interpretations, (two-valued) models, and stable models coincide. It hence follows (from the results in [SW15]) that acceptance problems can be decided in polynomial time for this subclass of ADFs. Current ADF systems are not optimised for acyclic ADFs, the purpose of comparing the performance of the systems on acyclic vs. non-acyclic ADFs thus being to determine whether the systems used at the back-end of the ADF systems (QBF and ASP solvers) are nevertheless able to use the acyclic nature of ADFs in their favour.

In the experimental setup of [Kes17] our generator used in the experiments in [BDH$^+$17] is modified to take undirected rather than directed graphs as input. For generating an acyclic ADF a directed graph is hence first generated from the undirected graph by choosing a total order on the vertices at random. This total order is used to determine the direction of the links. For the non-acyclic ADFs, a probability controls whether an edge in the input graph will result in a symmetric link in the ADF (in the experiments reported on in [Kes17] a probability of 0.5 is used); in case of non-symmetric links the direction of the link is chosen at random. ADFs were generated using the 8 metro networks also used in [BDH$^+$17] as input (now undirected) graphs. Ten different acyclic ADFs, resp. non-acyclic ADFs, were generated for each input graph. Hence, in total 160 ADF instances were generated, 80 acyclic and 80 non-acyclic ADFs.

The combinations of ADF and back-end systems used in [Kes17] are as in [BDH$^+$17], except that now also the system goDIAMOND [SE17] (version 0.6.6)[8] is considered. This is the more recent[9] variant of DIAMOND, which has also been submitted to the second

[8]https://sourceforge.net/p/diamond-adf/code/ci/go/tree/go/

[9]The reason for not using the other more recent versions of DIAMOND, versions 3.0.x implemented in

international competition for computational models of argumentation (ICCMA 2017; with some specialised encodings for Dung's AFs). This system still relies on static ASP encodings but is now implemented in the programming language `Go`. The system `goDIAMOND` also translates ADFs with acceptance conditions as propositional formulas to ADFs with acceptance conditions as boolean functions for most of the harder reasoning problems; given the complexity of this task, the system (version 0.6.6) does not support ADFs having statements with more than 31 parents.

The results obtained in the experiments in [Kes17] for non-acyclic ADFs, which we concentrate on here, are consistent with those obtained in our experiments from [BDH$^+$17]. Thus in general, the performance for credulous reasoning for the admissible semantics is quite good for the ASP-based systems: `YADF` has mean running time of 2.1679 seconds and `DIAMOND` (0.9) mean running time of 17.641 seconds. The system `goDIAMOND` excels with a mean running time of 0.202 seconds. None of the ASP-based systems has any time-outs (600 seconds). `QADF`, on the other hand, has quite a few time-outs (35) and mean running time of 5.5060 seconds (disregarding instances on which there were time-outs). All of the systems show improvement on the acyclic instances, the improvement being most notable for `QADF` whose time-outs are reduced to 4 instances (mean running time of 3.1635 seconds).

For the preferred semantics, as in the experiments in [BDH$^+$17], both `DIAMOND` and `QADF` time-out on all instances. Here the performance also of `YADF` is rather disappointing with 40 time-outs and mean running time of 126.128 seconds. The system `goDIAMOND`, on the other hand, has only 8 time-outs and mean running time of 1.2838 seconds. This is also the only system which showed improvement on the acyclic instances, having 0 time-outs and 0.144 mean running time on these instances.

Turning to the experiments reported on in [DKLW18], here also the performance of `goDIAMOND` (0.6.6), `YADF` (0.1.0), and `QADF` (0.3.2) (+ solver combinations as in [BDH$^+$17] and [Kes17]) are considered on acyclic vs non-acyclic ADFs produced by the generator as modified for [Kes17]. The difference is that now the benchmark set is generated from Dung argumentation frameworks interpreted as (undirected) graphs obtained from benchmarks used at the second international competition of argumentation (ICCMA'17) [AGLMW18]. These result from encoding assumption-based argumentation problems into AFs ("ABA") [LWJ17], encoding planning problems as AFs ("Planning") [CGV17], and a data-set of AFs generated from traffic networks ("Traffic"). This last data-set has, in fact, been submitted to ICCMA'17 by us [Dil17] and has been constructed using a larger set of traffic-networks from the same source we used for our evaluations of ADF systems in [BDH$^+$17]. Specifically, based on preliminary experiments, for our experiments in [DKLW18] we selected 100 AFs at random from a subset of AFs having up to 150 arguments in the very dense AFs in the "ABA" data-set, and 100 AFs at random from a subset of AFs having up to 300 arguments in each of the "Planning" and "Traffic" benchmarks. From the resulting 300 AFs interpreted as undirected graphs, we generated 300 acyclic and 300 non-acyclic ADFs.

`C++` [ES16], is the decrease of performance to previous versions of `DIAMOND` documented in [SE17].

Concentrating again on the results on the non-acyclic ADF instances, the results of our experiments for [DKLW18] are quite in line with those obtained in [Kes17] for credulous reasoning w.r.t. the admissible semantics for the "Traffic" and "Planning" instances. Nevertheless, all of the systems have somewhat more difficulty with the "Planning" (in relation to the "Traffic") instances; thus `goDIAMOND` and `YADF` have 0 time-outs (600 seconds) as well as 7.679 and 13.20 seconds mean running times respectively. `QADF` has 59 time-outs and 14.63 seconds mean running time on the instances on which it did not time-out. More novel results are obtained on the "ABA" data-set where there are many time-outs for all systems but, interestingly, `QADF` performs better with 30 time-outs and mean running time of 8.15 seconds. On the other hand, `goDIAMOND` has 52 time-outs and 22.64 seconds mean running time while `YADF` has 56 time-outs with 31.39 seconds mean running time. For the ASP-based systems there is only a slight improvement on their performance on the acyclic instances, while the increase in performance of `QADF` is (again, in line with the experiments for [Kes17]) more significant (114 time-outs on all non-acyclic vs 36 time-outs on all acyclic instances).

Also the results for the preferred semantics we report on in [DKLW18] are consistent with those in [Kes17] for the "Traffic" and "Planning" instances. Thus, for the (non-acyclic) "Traffic" instances `YADF` and `QADF` have 36 and 80 time-outs, while `goDIAMOND` has 3 time-outs and 11.12 seconds mean running time on the instances it did not time-out on. For the "Planning" instances `YADF` and `QADF` have 71 and 100 (i.e. time-out on all instances) time-outs, while `goDIAMOND` has 0 time-outs and 17.03 seconds mean running time. Again for the "ABA" instances the results are more novel, with all systems having many time-outs: 52, 57, and 81 for `goDIAMOND`, `YADF`, and `QADF` respectively. Although there are some improvements for some data-sets (mainly in the running times), no system performs much better (because no substantial improvement in the time-outs) on the acyclic instances of any of the data-sets for the preferred semantics.

Finally, we turn to considering the experiments reported on in [LMN$^+$18a]. In this study, the authors generate reasoning problems from the same set of ADFs we used for [DKLW18][10] but also consider a new ADF system they implement. Concretely, this study incorporates the incremental SAT based system `k++ADF`[11] which implements several link information sensitive as well as non-link-information sensitive SAT-based solving procedures for ADFs. This evaluation strategy is in line with our QBF encodings presented in [DWW14, DWW15] (non link information sensitive) and Section 3.1.3 of this work (link information sensitive) but where e.g. for the link information sensitive procedure the computation of the link types and solving are handled by subsequent calls to a SAT solver rather than in a monolithic encoding (as is the case for our encodings to QBF presented in Section 3.1.3).

For the study reported on in [LMN$^+$18a] the authors also make use of novel versions for the back-end systems w.r.t. previous experiments. Thus for `goDIAMOND` version 0.6.6 is

[10]The exact encodings used always differ in the different studies we consider in this appendix, because of the ways in which the statements whose acceptability is checked, is generated.

[11]`https://www.cs.helsinki.fi/group/coreo/k++adf/`

still used but now with `clingo` 5.2.1. For `QADF` version 0.3.2[12] with `bloqqer` 037 and `DepQBF` 6.03 is considered. Finally, for `YADF` version 0.1.0 is taken in account, but now with `lpopt` 2.2 and `clingo` 5.2.1. The version of `k++ADF` was presumably[13] version 2018-07-06; the SAT solver used is `MiniSAT` [ES03] version 2.2.0. For the experiments reported on in [LMN+18a] the time-out is also larger than for previous experiments: 1800 seconds.

Considering the differences in solvers used at the back-end as well as larger time-outs used in the study reported on in [LMN+18a] the results for `goDIAMOND` and `QADF` are in line with those in [DKLW18], except that notably more time-outs are reported on for `QADF` for the admissible semantics for the "ABA" and "Planning" data-sets (55 and 87 respectively vs 30 and 59 respectively in the experiments from [DKLW18]). For `YADF` on the other hand there are discrepancies, in particular for the admissible semantics. E.g. `YADF` has 47 and 21 time-outs on the "Traffic" and "Planning" data-sets, while in the study we reported on in [DKLW18] the time-outs were 2 and 0 respectively. We have determined[14] the cause of this to be the use of `lpopt` version 2.2, which seems to have problems in generating the rule-decompositions of some of the encodings obtained via `YADF` in a timely-manner; while versions previous to 2.2. (we also tried 2.0 and 2.1) don't have this issue.

Otherwise, clearly the novelty of the study from [LMN+18a] are the promising results for several variants of the incremental SAT-based approach. Thus, for the admissible semantics the link information sensitive algorithm implemented as part of `k++ADF` (`ADM-K-BIP`) has 0 time outs on the "Planning" and "Traffic" instances with 0.14 and 0.05 mean running times respectively. For the "ABA" data-set the non link information approach (`ADM-2`) has slightly less time-outs (11 rather than 15) than the link information sensitive approach and mean running time of 16.12 seconds. For the preferred semantics, the best performing variant of the incremental SAT approaches included in `k++ADF` is the link information sensitive procedure dubbed `PRF-K-BIP-OPT` which incorporates use of credulous reasoning for the admissible semantics and computation of the grounded interpretation as shortcuts to decide the skeptical acceptance problem. For the "Traffic" data set `PRF-K-BIP-OPT` has 1 time-out and 17.18 seconds mean running time, for the "Planning" data-set 3 time-outs and 11.14 seconds mean running time, while for the "ABA" data-set there are 16 time-outs and 25.90 seconds mean running time.

We end our survey of recent evaluations of ADF systems, including our own, by giving a bullet point summary of the results obtained so far. We focus on the experiments on data-sets resulting from generating ADFs based on the ICCMA'17 benchmarks (using the generator from [BDH+17] as later modified for [Kes17] and also taken up in later

[12]The paper mistakenly reports use of version 2.9.3 which does not exist (`QADF` version 0.3.2 is implemented in version 2.9.3 of the programming language `Scala`).

[13]This is not reported in the paper.

[14]Via tests carried out using the different versions of `lpopt` on instances for which there were time-outs in the study from [LMN+18a]. The instances were kindly made available to us by the authors of the latter work.

experiments) since, as we have detailed here, these expand while largely confirming the results of more constrained previous experiments on which they build ([BDH+17] and [Kes17]). In the summary, when mentioning results for a particular solver we use the best of the results for that solver obtained in the different studies.

As a reminder to the reader, in this discussion when we refer to the solvers `k++ADF`, `goDIAMOND`, `YADF`, and `QADF`, except if stated otherwise, these are versions 2018-07-06, 0.6.6, 0.1.0, and 0.3.2 respectively. In particular, we again note that `YADF` version 0.1.0 and `QADF` version 0.3.2 are identical to the newer versions detailed in this work (0.1.1 and 0.4.0; sections 3.2.7 and 3.1.4) for the encodings considered in the experiments we allude to.

- For credulous reasoning w.r.t. the admissible semantics each of `k++ADF` (when using the link information sensitive variant `ADM-K-BIP`, rather than `ADM-2`), `goDIAMOND`, and `YADF` (making use of `lpopt` version 2.0 as in the experiments from [BDH+17, Kes17, DKLW18]), have rather acceptable performance on the "Traffic" and "Planning" data-sets. (The same holds for `DIAMOND` version 0.9 on a small set of ADFs generated from metro-networks [BDH+17, Kes17].) The order in which we mention the solvers reflects the improvement in performance, with `k++ADF` being the clear "winner". The system `QADF` (even in the configuration with `bloqqer` version 035 and `DepQBF` version 4.0 from [BDH+17, Kes17, DKLW18]) on the other hand already has quite a few time-outs on the "Traffic" and "Planning" instances. We remind the reader that the "Traffic" and "Planning" data-sets include ADFs with 10 to 300 statements resulting from the underlying graphs obtained from representing transportation-networks as AFs [Dil17] and encoding planning problems into AFs [CGV17] respectively.
  - Thus `k++ADF` (implementing `ADM-K-BIP`) had 0 time-outs (1800 seconds) and 0.05 seconds mean running time, `goDIAMOND` 0 time-outs and 6.42 seconds mean running time in the experiments reported on in [LMN+18a] on the "Traffic" data-set. `YADF` had 2 time-outs (600 seconds) and 5.68 seconds mean running time in the experiments reported on in [DKLW18]. The system `k++ADF` (implementing `ADM-K-BIP`) had 0 time-outs and 0.14 seconds mean running time, `goDIAMOND` 0 time-outs and 6.72 seconds mean running time in the experiments reported on in [LMN+18a] on the "Planning" data-set. `YADF` had 0 time-outs and 13.20 seconds mean running time in the experiments reported on in [DKLW18]. `QADF` had 25 time-outs and 2.15 seconds mean running time on the "Traffic" instances and 59 time-outs and 14.63 seconds mean running time on the "Planning" instances [DKLW18].
- For credulous reasoning w.r.t. the admissible semantics, but now on the "ABA" data-set; here all ADF systems have some time-outs, yet again the results for `k++ADF` are the most promising. We remind the reader that the "ABA" data set consists in 100 very dense ADFs having between 10 to 150 statements resulting from

the underlying graphs of encoding problems for assumption-based-argumentation frameworks to AF reasoning problems [LWJ17].

  - Thus `k++ADF` (now in the `ADM-2` variant) had 12 time-outs (1800 seconds) and mean running time of 16.12 seconds in the experiments of [LMN+18a]. Interestingly, for the "ABA" data-set `QADF` (with `bloqqer` version 035 and `DepQBF` version 4.0) gets "second-place" having 30 time-outs (600 seconds) and 8.15 seconds mean running time in the experiments from [DKLW18]. The system `goDIAMOND` has 52 time-outs and `YADF` 56 time-outs in the experiments from [DKLW18].

- For the preferred semantics, the performance of our systems (as well as version 0.9 of `DIAMOND` on ADFs resulting from traffic networks [BDH+17, Kes17]) worsens considerably on the "Traffic" and "Planning" problems (w.r.t. results for the admissible semantics), while `k++ADF` (particularly in the variant `PRF-K-BIB-OPT`, but not in the variant `PRF-3`) and `goDIAMOND` have much better performance.
  - Thus `YADF` (`lpopt` version 2.0) has 36 time-outs on the "Traffic" instances and 71 time-outs on the "Planning" instances in the study from [DKLW18]. `QADF` has 80 and 100 time-outs on the "Traffic" and "Planning" benchmarks (again, study from [DKLW18]). On the other hand, `goDIAMOND` has 0 time-outs on both data-sets with 28.42 seconds and 17.52 seconds mean running times on the "Traffic" and "Planning" instances respectively [LMN+18a]. The system `k++ADF` (in the `PRF-K-BIB-OPT` variant) manages having only 1 time-out on the "Traffic" instances and 3 on the "Planning" instances with 17.18 and 11.14 seconds mean running time respectively [LMN+18a].

- All ADF systems also have time-outs when solving skeptical acceptance w.r.t the preferred semantics on the "ABA" data-set, with `k++ADF` in the `PRF-K-BIB-OPT` variant having the least (16 time-outs [LMN+18a]) and `QADF` the most (81 time-outs [DKLW18]).
  - Thus `k++ADF` in the `PRF-K-BIB-OPT` variant has 16 time-outs and 25.90 seconds mean running time [LMN+18a], while `QADF` has 81 time-outs (with 32.73 seconds running time on the remaining instances) [DKLW18]. `YADF` has 57 time-outs and 39.46 seconds mean running time, while `goDIAMOND` has 52 time-outs and 27.67 seconds mean running time [DKLW18].

To conclude, while our experiments from [BDH+17] (on the instances obtained via the grid-based generator first used in [Ell12] and ADFs constructed from a limited set of the traffic networks used in subsequent experiments) suggested `YADF` to be the better performing of the then considered systems (including `DIAMOND` version 0.9 and `QADF` 0.3.2), this picture has changed with subsequent experiments ([Kes17, DKLW18, LMN+18a]) involving the new systems `goDIAMOND` and `k++ADF` as well as more (and larger) data-sets. In particular, the clearly overall best performing approach for ADF systems seems to be, at current moment, the incremental SAT-based approach implemented in the system k++ADF (despite the fact that even this system still has quite a few time-outs for the preferred semantics on the ABA data-set). But even just considering ASP-based systems, while competitive for the admissible semantics, YADF is clearly behind in performance w.r.t. goDIAMOND for the preferred semantics on the "Traffic" and "Planning" data-sets.

A glimmer of hope for our systems YADF and QADF is provided by the results on the ABA data-set (in the configurations from [DKLW18]). For the ASP-based systems, here the constraint built into goDIAMOND of not supporting ADFs with statements having more than 31 parents is reflected in the constant number of time-outs on all reasoning tasks (admissible and preferred) and for acyclic as well as non-acyclic instances (the latter in the experiments from [DKLW18]). On the other hand, while YADF still has many time-outs (in fact, a few more than goDIAMOND) there is some (slight) improvement on the acyclic instances, which suggests room for improvement. Although QADF is quite behind its natural competitor, the newer k++ADF (even in its non-link-information-sensitive variants), the performance on the "ABA" data-set for the admissible semantics is again noteworthy (in particular, the improvement for the acyclic instances shown in [DKLW18]). More generally, comparing the results of QADF and the other ADF systems on the different data-sets, may suggest that QADF is less affected by the density of ADFs than the number of statements while this seems to be the other way around for the alternative ADF systems. This suggests a potential niche that could be exploited in further developing and evaluating also the system QADF.

Finally, we note that the experiments reported on in Appendix A comparing the performance of several QSAT solvers and preprocessors on our link-information-sensitive QBF encodings implemented in our new version of QADF (0.4.0; presented in Section 3.1.4) vs the non-link-information-sensitive QBF encodings (also considering different subclasses of ADFs) do not challenge our summary significantly. In particular, there is no substantial improvement when using our link-information-sensitive encodings resulting from ADFs generated on the base of traffic networks vs. the non-link-information-sensitive encodings, except for some of the poorer performing QBF solvers when used on encodings for the admissible semantics. On the other hand, the results obtained via the best performing combination of QSAT and preprocessors (namely, the version of DepQBF presented at QBFEVAL'18, also in combination with Bloqqer) when applied on the encodings (both link information and non link information sensitive variants) for the admissible semantics do improve on the results obtained in previous experimental evaluations. Thus, here QADF has only 1 (out of 78) time-out on the non-link-information-sensitive encodings with mean running time under 50 seconds, while 8 time-outs (out of 312) are obtained on the link-information-sensitive encodings with mean running times under 110 seconds.

List of Figures

List of Tables

List of Algorithms

Bibliography

[AAB16] Latifa Al-Abdulkarim, Katie Atkinson, and Trevor J. M. Bench-Capon. A methodology for designing systems to reason with legal cases using abstract dialectical frameworks. *Artif. Intell. Law*, 24(1):1–49, 2016.

[AAC^{+}18] Weronika T. Adrian, Mario Alviano, Francesco Calimeri, Bernardo Cuteri, Carmine Dodaro, Wolfgang Faber, Davide Fuscà, Nicola Leone, Marco Manna, Simona Perri, Francesco Ricca, Pierfrancesco Veltri, and Jessica Zangari. The ASP system DLV: advancements and applications. *KI*, 32(2-3):177–179, 2018.

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach.* Cambridge University Press, 2009.

[ABG^{+}17] Katie Atkinson, Pietro Baroni, Massimiliano Giacomin, Anthony Hunter, Henry Prakken, Chris Reed, Guillermo Ricardo Simari, Matthias Thimm, and Serena Villata. Towards Artificial Argumentation. *AI Magazine*, 38(3):25–36, 2017.

[AC13] Ofer Arieli and Martin W. A. Caminada. A QBF-based formalization of abstract argumentation semantics. *J. Applied Logic*, 11(2):229–252, 2013.

[ace13a] ACE 6.7 Construction Rules. Technical report, 2013. Available at `http://attempto.ifi.uzh.ch/site/docs/ace_constructionrules.html`.

[ace13b] ACE 6.7 Interpretation Rules. Technical report, 2013. Available at `http://attempto.ifi.uzh.ch/site/docs/ace_interpretationrules.html`.

[AF15] Mario Alviano and Wolfgang Faber. Stable Model Semantics of Abstract Dialectical Frameworks Revisited: A Logic Programming Perspective. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 2684–2690. AAAI Press, 2015.

[AFL10] Mario Alviano, Wolfgang Faber, and Nicola Leone. Disjunctive ASP with functions: Decidable queries and effective computation. *TPLP*, 10(4-6):497–512, 2010.

[AGLMW18] Sarah A. Gaggl, Thomas Linsbichler, Marco Maratea, and Stefan Woltran. Summary Report of the Second International Competition on Computational Models of Argumentation. *AI Magazine*, 39:77–79, 12 2018.

[AZZ17] Vernon Asuncion, Yan Zhang, and Heng Zhang. Polynomially Bounded Logic Programs with Function Symbols: A New Decidable. In *Proceedings of the 31rst AAAI Conference on Artificial Intelligence (AAAI 2017)*, pages 1041–1047. AAAI Press, 2017.

[Bau18] Ringo Baumann. On the Nature of Argumentation Semantics: Existence and Uniqueness, Expressibility, and Replaceability. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 17. College Publications, February 2018.

[BB05] Patrick Blackburn and Johan Bos. *Representation and Inference for Natural Language: A First Course in Computational Semantics.* CSLI Publications, 2005.

[BCG18] Pietro Baroni, Martin Caminada, and Massimiliano Giacomin. Abstract Argumentation Frameworks And Their Semantics. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 4. College Publications, February 2018.

[BD04] Philippe Besnard and Sylvie Doutre. Checking the acceptability of a set of arguments. In *Proceedings of the 10th International Workshop on Non-Monotonic Reasoning (NMR 2004)*, pages 59–64, 2004.

[BD07] Trevor J. M. Bench-Capon and Paul E. Dunne. Argumentation in artificial intelligence. *Artif. Intell.*, 171(10-15):619–641, 2007.

[BDH$^+$] Gerhard Brewka, Martin Diller, Georg Heissenberger, Thomas Linsbichler, and Stefan Woltran. Solving Advanced Argumentation Problems with Answer-Set Programming. Submitted to TPLP - Special issue on argumentation and logic programming - frontiers and relations.

[BDH$^+$17] Gerhard Brewka, Martin Diller, Georg Heissenberger, Thomas Linsbichler, and Stefan Woltran. Solving Advanced Argumentation Problems with Answer-Set Programming. In Satinder P. Singh and Shaul Markovitch, editors, *Proceedings of the 31rst Conference on Artificial Intelligence (AAAI 2017)*, pages 1077–1083. AAAI Press, 2017.

[BDKT97] Andrei Bondarenko, Phan Minh Dung, Robert A. Kowalski, and Francesca Toni. An Abstract, Argumentation-Theoretic Approach to Default Reasoning. *Artif. Intell.*, 93:63–101, 1997.

[Ber16] Matti Berthold. Extending the DIAMOND system to work with GRAPPA. In Matthias Thimm, Federico Cerutti, Hannes Strass, and Mauro Vallati, editors, *Proceedings of the 1rst International Workshop on Systems and Algorithms for Formal Argumentation (SAFA) co-located with the 6th International Conference on Computational Models of Argument (COMMA 2016)*, volume 1672 of *CEUR Workshop Proceedings*, pages 52–62. CEUR-WS.org, 2016.

[BES$^+$13] Gerhard Brewka, Stefan Ellmauthaler, Hannes Strass, Johannes P. Wallner, and Stefan Woltran. Abstract Dialectical Frameworks Revisited. In Francesca Rossi, editor, *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 803–809. AAAI Press / IJCAI, 2013.

[BES$^+$18] Gerhard Brewka, Stefan Ellmauthaler, Hannes Strass, Johannes P. Wallner, and Stefan Woltran. Abstract Dialectical Frameworks: An Overview. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 5. College Publications, February 2018.

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

[BET16] Gerhard Brewka, Thomas Eiter, and Miroslaw Truszczynski. Answer set programming: An introduction to the special issue. *AI Magazine*, 37(3):5–6, 2016.

[BH01] Philippe Besnard and Anthony Hunter. A logic-based theory of deductive arguments. *Artif. Intell.*, 128(1-2):203–235, 2001.

[BH18] Philippe Besnard and Anthony Hunter. A Review of Argumentation Based on Deductive Arguments. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 9. College Publications, February 2018.

[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.

[BMW16a] Manuel Bichler, Michael Morak, and Stefan Woltran. lpopt: A Rule Optimization Tool for Answer Set Programming. In *Proceedings of the 26th International Symposium on Logic-Based Program Synthesis and Transformation (LOPSTR 2016)*, volume 10184 of *Lecture Notes in Computer Science*, pages 114–130. Springer, 2016.

[BMW16b] Manuel Bichler, Michael Morak, and Stefan Woltran. The power of non-ground rules in answer set programming. *TPLP*, 16(5-6):552–569, 2016.

[BNT08] Gerhard Brewka, Ilkka Niemelä, and Miroslaw Truszczynski. Nonmonotonic Reasoning. In *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 239–284. Elsevier, 2008.

[BPW14] Gerhard Brewka, Sylwia Polberg, and Stefan Woltran. Generalizations of dung frameworks and their role in formal argumentation. *IEEE Intelligent Systems*, 29(1):30–38, 2014.

[Bra14] Max Bramer. *Logic Programming with Prolog*. Springer Publishing Company, Incorporated, 2nd edition, 2014.

[BSWW18] Gerhard Brewka, Hannes Strass, Johannes Peter Wallner, and Stefan Woltran. Weighted Abstract Dialectical Frameworks. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence, (AAAI 2018)*, pages 1779–1786. AAAI Press, 2018.

[BW10] Gerhard Brewka and Stefan Woltran. Abstract Dialectical Frameworks. In Fangzhen Lin, Ulrike Sattler, and Miroslaw Truszczyński, editors, *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning (KR 2010)*, pages 102–111. AAAI Press, 2010.

[BW14] Gerhard Brewka and Stefan Woltran. GRAPPA: A Semantical Framework for Graph-Based Argument Processing. In Torsten Schaub, Gerhard Friedrich, and Barry O'Sullivan, editors, *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 153–158. IOS Press, 2014.

[CA07] Martin Caminada and Leila Amgoud. On the evaluation of argumentation formalisms. *Artif. Intell.*, 171(5-6):286–310, 2007.

[Cam18] Martin Caminada. Rationality Postulates: applying argumentation-theory for non-monotonic reasoning. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 15. College Publications, February 2018.

[CDG$^+$15] Günther Charwat, Wolfgang Dvorák, Sarah Alice Gaggl, Johannes Peter Wallner, and Stefan Woltran. Methods for solving reasoning problems in abstract argumentation - A survey. *Artif. Intell.*, 220:28–63, 2015.

[CFST18] Kristijonas Cyras, Xiuyi Fan, Claudia Schulz, and Francesca Toni. Assumption-based argumentation: Disputes, explanations, preferences. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 7. College Publications, February 2018.

[CGTW18] Federico Cerutti, Sarah A. Gaggl, Matthias Thimm, and Johannes P. Wallner. Foundations of Implementations. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 14. College Publications, February 2018.

[CGV14] Federico Cerutti, Massimiliano Giacomin, and Mauro Vallati. ArgSemSAT: Solving Argumentation Problems Using SAT. In *Proceedings of the 5th International Conference on Computational Models of Argument (COMMA 2014)*, volume 266 of *Frontiers in Artificial Intelligence and Applications*, pages 455–456. IOS Press, 2014.

[CGV17] Federico Cerutti, Massimiliano Giacomin, and Mauro Vallati. Exploiting Planning Problems for Generating Challenging Abstract Argumentation Frameworks. `http://argumentationcompetition.org/2017/Planning2AF.pdf`, 2017.

[Coo71] S. Cook. The Complexity of Theorem-Proving Procedures. In *Conference Record of the 3rd Annual ACM Symposium on Theory of Computing (STOC 1971)*, pages 151–158, 1971.

[CTO14] Federico Cerutti, Nava Tintarev, and Nir Oren. Formal Arguments, Preferences, and Natural Language Interfaces to Humans: an Empirical Evaluation. In *Proceedings of the 21st European Conference on Artificial Intelligence (ECAI 2014)*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 207–212. IOS Press, 2014.

[CV16] Elena Cabrio and Serena Villata. Abstract Dialectical Frameworks for Text Exploration. In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016)*, pages 85–95. SciTePress, 2016.

[CV18] Elena Cabrio and Serena Villata. Five Years of Argument Mining: a Data-driven Analysis. In Jérôme Lang, editor, *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 5427–5433. ijcai.org, 2018.

[CVG16] Federico Cerutti, Mauro Vallati, and Massimiliano Giacomin. jArgSemSAT: An Efficient Off-the-Shelf Solver for Abstract Argumentation Frameworks. In *Proceedings of the 15th International Conference on Principles of Knowledge Representation and Reasoning (KR 2016)*, pages 541–544. AAAI Press, 2016.

[CVG17] Federico Cerutti, Mauro Vallati, and Massimiliano Giacomin. An Efficient Java-Based Solver for Abstract Argumentation Frameworks: jArgSemSAT. *International Journal on Artificial Intelligence Tools*, 26(2):1–26, 2017.

[CW17] Günther Charwat and Stefan Woltran. Expansion-based QBF Solving on Tree Decompositions. In Marco Maratea and Ivan Serina, editors, *Proceedings of the 24th RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion 2017 co-located with the 16th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2017)*, volume 2011 of *CEUR Workshop Proceedings*, pages 16–26. CEUR-WS.org, 2017.

[DB02] Paul E. Dunne and Trevor J. M. Bench-Capon. Coherence in finite argument systems. *Artif. Intell.*, 141(1/2):187–203, 2002.

[DB03] Paul E. Dunne and Trevor J. M. Bench-Capon. Two party immediate response disputes: Properties and efficiency. *Artif. Intell.*, 149(2):221–250, 2003.

[DD18] Wolfgang Dvořák and Paul E. Dunne. Computational Problems in Formal Argumentation and their Complexity. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 13. College Publications, February 2018.

[DDP$^+$18] Martin Diller, Wolfgang Dvořák, Jörg Pührer, Johannes Peter Wallner, and Stefan Woltran. Applications of ASP in Formal Argumentation. In *Proceedings of the 2nd Workshop on Trends and Applications of Answer Set Programming (TAASP 2018)*, 2018. Available online at `http://www.kr.tuwien.ac.at/events/taasp18/papers/TAASP_2018_paper_16.pdf`.

[DH16] Martin Diller and Anthony Hunter. Encoding monotonic multi-set preferences using CI-nets: preliminary report. *CoRR*, abs/1611.02885, 2016.

[DH17] Martin Diller and Anthony Hunter. Encoding monotonic multiset preferences using CI-nets. In Bernhard Mitschang, Norbert Ritter, Holger Schwarz, Meike Klettke, Andreas Thor, Oliver Kopp, and Matthias Wieland, editors, *Proceedings of the 1rst Workshop on Preferences and Personalisation in Informatics (PPI 2017) at the 17th International Conference on Database Systems for Business, Technology, and Web of the German Informatics Society (BTW 2017)*, volume P-266 of *LNI*, pages 169–180. GI, 2017.

[DHL$^+$15] Martin Diller, Adrian Haret, Thomas Linsbichler, Stefan Rümmele, and Stefan Woltran. An Extension-Based Approach to Belief Revision in Abstract Argumentation. In Qiang Yang and Michael Wooldridge, editors, *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 2926–2932. AAAI Press, 2015.

[DHL$^+$18] Martin Diller, Adrian Haret, Thomas Linsbichler, Stefan Rümmele, and Stefan Woltran. An Extension-Based Approach to Belief Revision in Abstract Argumentation. *Int. J. Approx. Reasoning*, 93:395–423, 2018.

[Dil14] Martin Diller. Solving Reasoning Problems on Abstract Dialectical Frameworks via Quantified Boolean Formulas. Master's thesis, Vienna University of Technology, Institute of Information Systems, 2014.

[Dil17] Martin Diller. Traffic Networks Become Argumentation Frameworks. `http://argumentationcompetition.org/2017/Traffic.pdf`, 2017.

[DJWW14] Wolfgang Dvorák, Matti Järvisalo, Johannes Peter Wallner, and Stefan Woltran. Complexity-sensitive decision procedures for abstract argumentation. *Artif. Intell.*, 206:53–78, 2014.

[DKLW18] Martin Diller, Atefeh Keshavarzi Zafarghandi, Thomas Linsbichler, and Stefan Woltran. Investigating Subclasses of Abstract Dialectical Frameworks. In *Proceedings of the 7th International Conference on Computational Models of Argument COMMA 2018*, volume 305 of *Frontiers in Artificial Intelligence and Applications*, pages 61–72. IOS Press, 2018.

[DT18] Phan Minh Dung and Phan Minh Thang. Representing the semantics of abstract dialectical frameworks based on arguments and attacks. *Argument & Computation*, 9(3):249–267, 2018.

[Dun95] Phan Minh Dung. On the Acceptability of Arguments and its Fundamental Role in Nonmonotonic Reasoning, Logic Programming and N-person Games. *Artificial Intelligence*, 77(2):321–357, 1995.

[DWS17] Martin Diller, Adam Wyner, and Hannes Strass. Defeasible AceRules: A prototype. In Claire Gardent and Christian Retoré, editors, *Proceedings of the 12th International Conference on Computational Semantics (IWCS 2017)*, September 2017.

[DWS19] Martin Diller, Adam Wyner, and Hannes Strass. Making sense of conflicting (defeasible) rules in the controlled natural language ACE: design of a system with support for existential quantification using skolemization. In *Proceedings of the 13th International Conference on Computational Semantics (IWCS 2019)*, 2019. To appear.

[DWW14] Martin Diller, Johannes Peter Wallner, and Stefan Woltran. Reasoning in Abstract Dialectical Frameworks Using Quantified Boolean Formulas. In *Proceedings of the 5th International Conference on Computational Models of Argument (COMMA 2014)*, volume 266 of *Frontiers in Artificial Intelligence and Applications*, pages 241–252. IOS Press, 2014.

[DWW15] Martin Diller, Johannes Peter Wallner, and Stefan Woltran. Reasoning in abstract dialectical frameworks using quantified Boolean formulas. *Argument & Computation*, 6(2):149–177, 2015.

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

[EG95] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.*, 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.

[EGW10] Uwe Egly, Sarah Alice Gaggl, and Stefan Woltran. Answer-set programming encodings for argumentation frameworks. *Argument & Computation*, 1(2):147–177, 2010.

[EIK09] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. Answer set programming: A primer. In *Reasoning Web*, volume 5689 of *Lecture Notes in Computer Science*, pages 40–110. Springer, 2009.

[Ell12] Stefan Ellmauthaler. Abstract Dialectical Frameworks: Properties, Complexity, and Implementation. Master's thesis, Vienna University of Technology, Institute of Information Systems, 2012.

[End72] Herbert B. Enderton. *A Mathematical Introduction to Logic.* 1972.

[EÖ15] Esra Erdem and Umut Öztok. Generating explanations for biomedical queries. *TPLP*, 15(1):35–78, 2015.

[ES03] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing (SAT 2003)*, volume 2919 of *Lecture Notes in Computer Science*, pages 502–518. Springer, 2003.

[ES14] Stefan Ellmauthaler and Hannes Strass. The DIAMOND system for computing with abstract dialectical frameworks. In *Proceedings of the 5th International Conference on Computational Models of Argument (COMMA 2014)*, volume 266 of *Frontiers in Artificial Intelligence and Applications*, pages 233–240. IOS Press, 2014.

[ES16] Stefan Ellmauthaler and Hannes Strass. DIAMOND 3.0 - A native C++ implementation of DIAMOND. In Pietro Baroni, Thomas F. Gordon, Tatjana Scheffler, and Manfred Stede, editors, *Proceedings of the 6th*

International Conference on Computational Models of Argument (COMMA 2016), volume 287 of *Frontiers in Artificial Intelligence and Applications*, pages 471–472. IOS Press, 2016.

[EW06] Uwe Egly and Stefan Woltran. Reasoning in Argumentation Frameworks Using Quantified Boolean Formulas. In *Proceedings of the 1rst International Conference on Computational Models of Argument (COMMA 2006)*, volume 144 of *Frontiers in Artificial Intelligence and Applications*, pages 133–144. IOS Press, 2006.

[EW12] Stefan Ellmauthaler and Johannes Peter Wallner. Evaluating Abstract Dialectical Frameworks with ASP. In *Proceedings of the 4th International Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 505–506. IOS Press, 2012.

[FKK08] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Attempto Controlled English for knowledge representation. In *Tutorial lectures of the 4th International Summer School on the Reasoning Web (Reasoning Web 2008)*, pages 104–124, 2008.

[FKK13] Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Discourse Representation Structures for ACE 6.7. Technical report, 2013. Available at `http://attempto.ifi.uzh.ch/site/pubs/papers/drs_report_67.pdf`.

[Fuc10] Norbert E. Fuchs. First-Order Reasoning for Attempto Controlled English. In Michael Rosner and Norbert E. Fuchs, editors, *Proceedings of the 2nd International Workshop on Controlled Natural Language (CNL 2010)*, volume 7175 of *Lecture Notes in Computer Science*, pages 73–94. Springer, 2010.

[FVCG18] Wolfgang Faber, Mauro Vallati, Federico Cerutti, and Massimiliano Giacomin. Enumerating Preferred Extensions Using ASP Domain Heuristics: The ASPrMin Solver. In *Proceedings of the 7th International Conference on Computational Models of Argument (COMMA 2018)*, volume 305 of *Frontiers in Artificial Intelligence and Applications*, pages 459–460. IOS Press, 2018.

[GGLS15] Fabien Garreau, Laurent Garcia, Claire Lefèvre, and Igor Stéphan. ∃-ASP. In *Proceedings of the Joint Ontology Workshops 2015 Episode 1: The Argentine Winter of Ontology co-located with the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, volume 1517 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2015.

[GHVD03] Benjamin N. Grosof, Ian Horrocks, Raphael Volz, and Stefan Decker. Description logic programs: combining logic programs with description logic. In Gusztáv Hencsey, Bebo White, Yih-Farn Robin Chen, László Kovács, and Steve Lawrence, editors, *Proceedings of the 12th International World Wide Web Conference, (WWW 2003)*, pages 48–57. ACM, 2003.

[GKK$^+$18] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Patrick Lühne, Philipp Obermeier, Max Ostrowski, Javier Romero, Torsten Schaub, Sebastian Schellhorn, and Philipp Wanko. The Potsdam Answer Set Solving Collection 5.0. *KI*, 32(2-3):181–182, 2018.

[GKKS12] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. *Answer Set Solving in Practice.* Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[GL88] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP*, pages 1070–1080. MIT Press, 1988.

[GL90] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In *ICLP*, pages 579–597. MIT Press, 1990.

[GL91] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Comput.*, 9(3/4):365–386, 1991.

[GLMW16] Sarah Alice Gaggl, Thomas Linsbichler, Marco Maratea, and Stefan Woltran. Introducing the Second International Competition on Computational Models of Argumentation. In *Proceedings of the 1rst International Workshop on Systems and Algorithms for Formal Argumentation (SAFA) co-located with the 6th International Conference on Computational Models of Argument (COMMA 2016)*, volume 1672 of *CEUR Workshop Proceedings*, pages 4–9. CEUR-WS.org, 2016.

[Gro97] Benjamin N. Grosof. Prioritized Conflict Handling for Logic Programs. In Jan Maluszynski, editor, *Proceedings of the 1997 International Symposium on Logic Programming*, pages 197–211. MIT Press, 1997.

[Gro17] Leo Groarke. Informal logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy.* Metaphysics Research Lab, Stanford University, spring 2017 edition, 2017.

[GS04] Alejandro J. García and Guillermo R. Simari. Defeasible Logic Programming: An Argumentative Approach. *TPLP*, 4(1-2):95–138, 2004.

[GS16] Martin Gebser and Torsten Schaub. Modeling and Language Extensions. *AI Magazine*, 37(3):33–44, 2016.

[GS17] Stephen Guy and Rolf Schwitter. The PENG ASP system: architecture, language and authoring tool. *Language Resources and Evaluation*, 51(1):67–92, 2017.

[GS18] Alejandro J. Garcia and Guillermo R. Simari. Argumentation Based on Logic Programming. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 8. College Publications, February 2018.

[Hei16] Georg Heissenberger. A System For Advanced Graphical Argumentation Formalisms. Master's thesis, Vienna University of Technology, Institute of Information Systems, 2016.

[HJL$^+$15] Marijn Heule, Matti Järvisalo, Florian Lonsing, Martina Seidl, and Armin Biere. Clause elimination for SAT and QSAT. *J. Artif. Intell. Res.*, 53:127–168, 2015.

[HW12] Anthony Hunter and Matthew Williams. Aggregating evidence about the positive and negative effects of treatments. *Artificial Intelligence in Medicine*, 56(3):173–190, 2012.

[HW16] Georg Heissenberger and Stefan Woltran. GrappaVis - A System for Advanced Graph-Based Argumentation. In Pietro Baroni, Thomas F. Gordon, Tatjana Scheffler, and Manfred Stede, editors, *Proceedings of the 6th International Conference on Computational Models of Argument (COMMA 2016)*, volume 287 of *Frontiers in Artificial Intelligence and Applications*, pages 473–474. IOS Press, 2016.

[JJK$^+$14] Mikolas Janota, Charles Jordan, Will Klieber, Florian Lonsing, Martina Seidl, and Allen Van Gelder. The QBFGallery 2014: The QBF Competition at the FLoC Olympic Games. *JSAT*, 9:187–206, 2014.

[JKMC16] Mikolás Janota, William Klieber, Joao Marques-Silva, and Edmund M. Clarke. Solving QBF with counterexample guided refinement. *Artif. Intell.*, 234:1–25, 2016.

[JKS16] Charles Jordan, Will Klieber, and Martina Seidl. Non-cnf QBF solving with QCIR. In *AAAI Workshop: Beyond NP*, volume WS-16-05 of *AAAI Workshops*. AAAI Press, 2016.

[JN16] Tomi Janhunen and Ilkka Niemelä. The Answer Set Programming Paradigm. *AI Magazine*, 37(3):13–24, 2016.

[Kal09] Kaarel Kaljurand. Paraphrasing Controlled English Texts. In *Pre-Proceedings of the Workshop on Controlled Natural Language (CNL 2009)*, volume 448 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[KB09] Hans Kleine Büning and Uwe Bubeck. Theory of Quantified Boolean Formulas. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 735–760. IOS Press, 2009.

[Kes17] Atefeh Keshavarzi Zafarghandi. Investigating Subclasses ofAbstract Dialectical Frameworks. Master's thesis, Vienna University of Technology, Institute of Information Systems, 2017.

[Kuh07] Tobias Kuhn. AceRules: Executing Rules in Controlled Natural Language. In *Proceedings of the 1rst International Conference on Web Reasoning and Rule Systems (RR 2007)*, volume 4524 of *Lecture Notes in Computer Science*, pages 299–308. Springer, 2007.

[Kuh09] Tobias Kuhn. AceWiki: A Semantic Wiki Using Controlled English. In *Proceedings of the Poster Session at the 6th European Semantic Web Conference (ESWC 2009)*, 2009.

[Kuh10] Tobias Kuhn. *Controlled English for Knowledge Representation.* PhD thesis, University of Zurich, 2010.

[Kuh14] Tobias Kuhn. A Survey and Classification of Controlled Natural Languages. *Computational Linguistics*, 40(1):121–170, March 2014.

[LB10] Florian Lonsing and Armin Biere. DepQBF: A Dependency-Aware QBF Solver. *JSAT*, 7(2-3):71–76, 2010.

[LBSG17] Claire Lefèvre, Christopher Béatrix, Igor Stéphan, and Laurent Garcia. ASPeRiX, a first-order forward chaining approach for answer set computing. *TPLP*, 17(3):266–310, 2017.

[LE17] Florian Lonsing and Uwe Egly. DepQBF 6.0: A Search-Based QBF Solver Beyond Traditional QCDCL. In *Proceedings of the 26th International Conference on Automated Deduction (CADE 2017)*, volume 10395 of *Lecture Notes in Computer Science*, pages 371–384. Springer, 2017.

[LE18a] Florian Lonsing and Uwe Egly. Evaluating QBF Solvers: Quantifier Alternations Matter. In John N. Hooker, editor, *Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming (CP 2018)*, volume 11008 of *Lecture Notes in Computer Science*, pages 276–294. Springer, 2018.

[LE18b] Florian Lonsing and Uwe Egly. QRAT+: Generalizing QRAT by a More Powerful QBF Redundancy Property. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Proceedings of the 9th International Joint Conference on Automated Reasoning (IJCAR 2018)*, volume 10900 of *Lecture Notes in Computer Science*, pages 161–177. Springer, 2018.

[Lev73] L. A. Levin. Universal sequential search problems. *Problems of Information Transmission*, 9(3):265–266, 1973.

[LL13] Yuliya Lierler and Vladimir Lifschitz. Logic Programs vs. First-Order Formulas in Textual Inference. In Katrin Erk and Alexander Koller, editors, *Proceedings of the 10th International Conference on Computational Semantics (IWCS 2013)*, pages 340–346. The Association for Computer Linguistics, 2013.

[LMN$^+$18a] Thomas Linsbichler, Marco Maratea, Andreas Niskanen, Johannes Peter Wallner, and Stefan Woltran. Novel Algorithms for Abstract Dialectical Frameworks based on Complexity Analysis of Subclasses and SAT Solving. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI 2018)*, pages 1905–1911. ijcai.org, 2018.

[LMN$^+$18b] Thomas Linsbichler, Marco Maratea, Andreas Niskanen, Johannes Peter Wallner, and Stefan Woltran. Novel algorithms for abstract dialectical frameworks based on complexity analysis of subclasses and SAT solving. In *IJCAI*, pages 1905–1911. ijcai.org, 2018.

[LPS16] Thomas Linsbichler, Jörg Pührer, and Hannes Strass. A uniform account of realizability in abstract argumentation. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)*, volume 285 of *Frontiers in Artificial Intelligence and Applications*, pages 252–260. IOS Press, 2016.

[LSG16] Florian Lonsing, Martina Seidl, and Allen Van Gelder. The QBF Gallery: Behind the scenes. *Artif. Intell.*, 237:92–114, 2016.

[LWJ17] Tuomo Lehtonen, Johannes P. Wallner, and Matti Järvisalo. Assumption-Based Argumentation Translated to Argumentation Frameworks. `http://argumentationcompetition.org/2017/ABA2AF.pdf`, 2017.

[MB88] Rainer Manthey and François Bry. SATCHMO: A Theorem Prover Implemented in Prolog. In Ewing L. Lusk and Ross A. Overbeek, editors, *Proceedings of the 9th International Conference on Automated Deduction (CADE 1988)*, volume 310 of *Lecture Notes in Computer Science*, pages 415–434. Springer, 1988.

[MNT11] Victor W. Marek, Ilkka Niemelä, and Mirosław Truszczyński. Origins of Answer-Set Programming – Some Background and Two Personal Accounts. In Gerhard Brewka, Victor W. Marek, and Mirosław Truszczyński, editors, *Nonmonotonic Reasoning: Essays Celebrating its 30th Anniversary*, pages 233–258. College Publications, London, UK, 2011.

[Mod13] Sanjay Modgil. Revisiting Abstract Argumentation Frameworks. In Elizabeth Black, Sanjay Modgil, and Nir Oren, editors, *Proceedings of the 2nd*

International Workshop on Theory and Applications of Formal Argumentation (TAFA 2013), volume 8306 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2013.

[MP09] Peter McBurney and Simon Parsons. Dialogue Games for Agent Argumentation. In Guillermo Ricardo Simari and Iyad Rahwan, editors, *Argumentation in Artificial Intelligence*, pages 261–280. Springer, 2009.

[MP18] Sanjay Modgil and Henry Prakken. Abstract Rule-Based Argumentation. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 6. College Publications, February 2018.

[MT99] Victor W. Marek and Miroslaw Truszczyński. Stable models and an alternative logic programming paradigm. In Krzysztof R. Apt, Victor W. Marek, Mirek Truszczynski, and David S. Warren, editors, *The Logic Programming Paradigm: A 25-Year Perspective*, pages 375–398. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

[NCO08] Juan Carlos Nieves, Ulises Cortés, and Mauricio Osorio. Preferred extensions as stable models. *TPLP*, 8(4):527–543, 2008.

[Neu18] Daniel Neugebauer. DABASCO: Generating AF, ADF, and ASPIC$^+$ Instances from Real-World Discussions. In Sanjay Modgil, Katarzyna Budzynska, and John Lawrence, editors, *Proceedings of the 7th International Conference on Computational Models of Argument (COMMA 2018)*, volume 305 of *Frontiers in Artificial Intelligence and Applications*, pages 469–470. IOS Press, 2018.

[Nie99] Ilkka Niemelä. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. *Ann. Math. Artif. Intell.*, 25(3-4):241–273, 1999.

[Pap07] Christos H. Papadimitriou. *Computational complexity.* Academic Internet Publ., 2007.

[Par90] T. Parsons. *Events in the Semantics of English: A Study of Subatomic Semantics.* MIT Press, Cambridge, MA, USA, 1s edition, 1990.

[PD14] Sylwia Polberg and Dragan Doder. Probabilistic Abstract Dialectical Frameworks. In *Proceedings of the 14th European Conference on Logics in Artificial Intelligence (JELIA 2014)*, volume 8761 of *Lecture Notes in Computer Science*, pages 591–599. Springer, 2014.

[PLB$^+$17] Alison Pease, John Lawrence, Katarzyna Budzynska, Joseph Corneli, and Chris Reed. Lakatos-style collaborative mathematics through dialectical, structured and abstract argumentation. *Artif. Intell.*, 246:181 – 219, 2017.

[PLJ17] Fuan Pu, Guiming Luo, and Zhou Jiang. Encoding Argumentation Semantics by Boolean Algebra. *IEICE Transactions*, 100-D(4):838–848, 2017.

[Pol14] Sylwia Polberg. Extension-Based Semantics of Abstract Dialectical Frameworks. In *Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS 2014)*, volume 264 of *Frontiers in Artificial Intelligence and Applications*, pages 240–249. IOS Press, 2014.

[Pol16] Sylwia Polberg. Understanding the abstract dialectical framework. In *Proceedings of the 15th European Conference on Logics in Artificial Intelligence (JELIA 2016)*, volume 10021 of *Lecture Notes in Computer Science*, pages 430–446, 2016.

[Pol17] Sylwia Polberg. *Developing the Abstract Dialectical Framework*. PhD thesis, Vienna University of Technology, 2017.

[Pra18] Henry Prakken. Historical Overview of Formal Argumentation. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 2. College Publications, February 2018.

[PW18] Henry Prakken and Michiel De Winter. Abstraction in Argumentation: Necessary but Dangerous. In Sanjay Modgil, Katarzyna Budzynska, and John Lawrence, editors, *Proceedings of the 7th International Conference on Computational Models of Argument (COMMA 2018)*, volume 305 of *Frontiers in Artificial Intelligence and Applications*, pages 85–96. IOS Press, 2018.

[Rei78] Raymond Reiter. On Reasoning by Default. In *Proceedings of the 1978 workshop on Theoretical issues in natural language processing*, pages 210–218, Morristown, NJ, USA, 1978. Association for Computational Linguistics.

[RT15] Markus N. Rabe and Leander Tentrup. CAQE: A certifying QBF solver. In *Proceedings of the 15h Conference on Formal Methods in Computer-Aided Design (FMCAD 2015)*, pages 136–143. IEEE, 2015.

[RvBW06] Francesca Rossi, Peter van Beek, and Toby Walsh, editors. *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*. Elsevier, 2006.

[Sch18a] Peter Schüller. Answer Set Programming in Linguistics. *KI*, 32(2-3):151–155, 2018.

[Sch18b] Rolf Schwitter. Specifying and Verbalising Answer Set Programs in Controlled Natural Language. *TPLP*, 18(3-4):691–705, 2018.

[SE17] Hannes Strass and Stefan Ellmauthaler. go DIAMOND 0.6.6 ICCMA 2017 System Description. `http://argumentationcompetition.org/2017/goDIAMOND.pdf`, 2017.

[SM73] Larry J. Stockmeyer and Albert R. Meyer. Word Problems Requiring Exponential Time: Preliminary Report. In *Proceedings of the 5th Annual ACM Symposium on Theory of Computing (STOC 1973)*, pages 1–9. ACM, 1973.

[Sto76] Larry J. Stockmeyer. The Polynomial-Time Hierarchy. *Theor. Comput. Sci.*, 3(1):1–22, 1976.

[Str13] Hannes Strass. Approximating operators and semantics for abstract dialectical frameworks. *Artif. Intell.*, 205:39–70, 2013.

[Str15a] Hannes Strass. Expressiveness of Two-Valued Semantics for Abstract Dialectical Frameworks. *J. Artif. Intell. Res.*, 54:193–231, 2015.

[Str15b] Hannes Strass. The Relative Expressiveness of Abstract Argumentation and Logic Programming. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI 2015)*, pages 1625–1631. AAAI Press, 2015.

[Str18] Hannes Strass. Instantiating rule-based defeasible theories in abstract dialectical frameworks and beyond. *J. Log. Comput.*, 28(3):605–627, 2018.

[SW15] Hannes Strass and Johannes Peter Wallner. Analyzing the computational complexity of abstract dialectical frameworks via approximation fixpoint theory. *Artif. Intell.*, 226:34–74, 2015.

[SW17] Hannes Strass and Adam Wyner. On Automated Defeasible Reasoning with Controlled Natural Language and Argumentation. In Roman Barták, Thomas Leo McCluskey, and Enrico Pontelli, editors, *Proceedings of the 2nd International Workshop on Knowledge-based Techniques for Problem Solving and Reasoning (KnowProS 2017)*, February 2017.

[SW18a] Torsten Schaub and Stefan Woltran. Answer set programming unleashed! *KI*, 32(2-3):105–108, 2018.

[SW18b] Torsten Schaub and Stefan Woltran. Special Issue on Answer Set Programming. *KI*, 32(2-3):101–103, 2018.

[SWD] Hannes Strass, Adam Wyner, and Martin Diller. EMIL: Extracting Meaning from Inconsistent Language. Towards argumentation using a controlled natural language interface. Submitted to International Journal of Approximate Reasoning.

[Tar55] A. Tarski. A Lattice-Theoretical Fixpoint Theorem and Its Applications. *Pacific Journal of Mathematics*, pages 285–309, 1955.

[Ten18] Leander Tentrup. On expansion and resolution in CEGAR based QBF solving. *CoRR*, abs/1803.09559, 2018.

[TS11] Francesca Toni and Marek Sergot. Argumentation and answer set programming. In Marcello Balduccini and Tran C. Son, editors, *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays in Honor of Michael Gelfond*, pages 164–180. Springer, 2011.

[Tse68] G. S. Tseitin. On the Complexity of Derivations in the Propositional Calculus. *Studies in Mathematics and Mathematical Logic*, Part II:115–125, 1968.

[Tur37] A. M. Turing. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265, 1937.

[TV17] Matthias Thimm and Serena Villata. The first international competition on computational models of argumentation: Results and analysis. *Artif. Intell.*, 252:267–294, 2017.

[TVC$^+$16] Matthias Thimm, Serena Villata, Federico Cerutti, Nir Oren, Hannes Strass, and Mauro Vallati. Summary Report of The First International Competition on Computational Models of Argumentation. *AI Magazine*, 37(1):102, 2016.

[vdTV18] Leender van der Torre and Srdjan Vesic. The Principle-Based Approach to Abstract-Argumentation Semantics. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 16. College Publications, February 2018.

[vEV18] Frans H. van Eemeren and Bart Verheij. Argumentation Theory in Formal and Computational Perspective. In Pietro Baroni, Dov Gabbay, Massimiliano Giacomin, and Leendert van der Torre, editors, *Handbook of Formal Argumentation*, chapter 1. College Publications, February 2018.

[vH02] Jean van Heijenoort. *From Frege to Gödel : A Source Book in Mathematical Logic, 1879-1931 (Source Books in the History of the Sciences).* Harvard University Press, January 2002.

[Wal14] Johannes Peter Waller. *Complexity Results and Algorithms for Argumentation - Dung's Frameworks and Beyond.* PhD thesis, Vienna University of Technology, 2014.

[WBDC15] Adam Z. Wyner, Trevor J. M. Bench-Capon, Paul E. Dunne, and Federico Cerutti. Senses of 'argument' in instantiated argumentation frameworks. *Argument & Computation*, 6(1):50–72, 2015.

[WN08] Toshiko Wakaki and Katsumi Nitta. Computing Argumentation Semantics in Answer Set Programming. In *Proceedings of the Conference on New Frontiers in Artificial Intelligence (JSAI 2008)*, volume 5447 of *Lecture Notes in Computer Science*, pages 254–269. Springer, 2008.

[Wra76] Celia Wrathall. Complete Sets and the Polynomial-Time Hierarchy. *Theor. Comput. Sci.*, 3(1):23–33, 1976.

[WRMB17] Ralf Wimmer, Sven Reimer, Paolo Marin, and Bernd Becker. HQSpre - An Effective Preprocessor for QBF and DQBF. In *Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2017)*, volume 10205 of *Lecture Notes in Computer Science*, pages 373–390, 2017.

[WS17] Adam Z. Wyner and Hannes Strass. dARe - Using Argumentation to Explain Conclusions from a Controlled Natural Language Knowledge Base. In *IEA/AIE (2)*, volume 10351 of *Lecture Notes in Computer Science*, pages 328–338. Springer, 2017.

[WSAB12] Adam Z. Wyner, Jodi Schneider, Katie Atkinson, and Trevor J. M. Bench-Capon. Semi-Automated Argumentative Analysis of Online Product Reviews. In *Proceedings of the 4th International Conference on Computational Models of Argument (COMMA 2012)*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 43–50. IOS Press, 2012.

[WvEH16] Adam Z. Wyner, Tom M. van Engers, and Anthony Hunter. Working on the argument pipeline: Through flow issues between natural language argument, instantiated arguments, and argumentation frameworks. *Argument & Computation*, 7(1):69–89, 2016.