Proceedings of the 22nd International Conference on Principles of Knowledge Representation and Reasoning
Main Track

Grounding Rule-Based Argumentation Using Datalog

Martin Diller[1] , **Sarah Alice Gaggl**[1] , **Philipp Hanisch**[2] , **Giuseppina Monterosso**[3] , **Fritz Rauschenbach**[1]
[1]Logic Programming and Argumentation Group, TU Dresden, Germany
[2]Knowledge-Based Systems Group, TU Dresden, Germany
[3]DIMES - University of Calabria, Italy
{martin.diller, sarah.gaggl, philipp.hanisch1}@tu-dresden.de, giusy.monterosso@dimes.unical.it, fritz.rauschenbach@proton.me

Abstract

ASPIC+ is one of the main general frameworks for rule-based argumentation for AI. Although first-order rules are commonly used in ASPIC+ examples, most existing approaches to reason over rule-based argumentation only support propositional rules. To enable reasoning over first-order instances, a preliminary grounding step is required. As groundings can lead to an exponential increase in the size of the input theories, intelligent procedures are needed. However, there is a lack of dedicated solutions for ASPIC+. Therefore, we propose an intelligent grounding procedure that keeps the size of the grounding manageable while preserving the correctness of the reasoning process. To this end, we translate the first-order ASPIC+ instance into a Datalog program and query a Datalog engine to obtain ground substitutions to perform the grounding of rules and contraries. Additionally, we propose simplifications specific to the ASPIC+ formalism to avoid grounding of rules that have no influence on the reasoning process. Finally, we performed an empirical evaluation of a prototypical implementation to show scalability.

1 Introduction

Rule-based argumentation formalisms such as ASPIC+ (Prakken 2010) and the closely related Assumption-based Argumentation (ABA) (Bondarenko, Toni, and Kowalski 1993; Bondarenko et al. 1997) provide a means to model and reason about complex scenarios involving conflicting information (see also (Besnard et al. 2014) for an overview). While ASPIC+ is a general abstract framework, a particularly relevant instance is what we refer to as (concrete) rule-based ASPIC+, also known as the logic-programming or Horn variant of ASPIC+. As the name suggests, in this variant, conflicts within knowledge bases composed of strict and defeasible Horn rules are resolved by inspecting arguments and counterarguments constructed using these rules. In particular, since rule-based ASPIC+ captures Answer Set Programming (ASP) (Marek and Truszczynski 1999; Niemelä 1999) under the stable semantics, it also offers an argument-based characterization of ASP (Modgil and Prakken 2018; Bondarenko et al. 1997). On the other hand, rule-based ASPIC+ supports a more expressive syntax than (normal) ASP—such as a generalized contrary relation and the inclusion of defeasible rules—and it relies on argumentation semantics (Dung 1995) rather than distinguished Herbrand models.

Although ASPIC+ and ASP share overlapping conceptual roots, ASPIC+ lags behind ASP in terms of the modelling constructs available in practical systems (see e.g. (Gebser and Schaub 2016)). In particular, a key advantage of ASP systems is their support for first-order variables in Horn rules, enabling concise representations through universal quantification—a crucial feature for knowledge representation and reasoning. It is similarly natural for ASPIC+ instances to make use of first-order logic-programming-like rules, which support more compact and general modeling (Modgil and Prakken 2018). Nevertheless, a concrete syntax for first-order ASPIC+ is often left unspecified. More critically, most existing computational techniques for reasoning in rule-based ASPIC+ assume that all rules are ground instances, i.e., propositional Horn rules (Lehtonen, Wallner, and Järvisalo 2020; Odekerken et al. 2023; Lehtonen et al. 2024a).

Relatedly, recent years have seen growing interest in developing sophisticated solvers for argumentation-exemplified by the International Competition on Computational Models of Argument (ICCMA), which has been held biennially since 2015 and in 2023 included, for the first time, a structured argumentation track focused on (flat) ABA frameworks (Järvisalo, Lehtonen, and Niskanen 2025), which can be captured in ASPIC+ (Modgil and Prakken 2018). Nevertheless, there remains a lack of benchmarks for rule-based instances derived from real-world problems. This gap may be partly due to the absence of effective methods for converting first-order rule knolwedge bases into their propositional counterparts.

For all these reasons, support for grounding—i.e., replacing variables with constants—is essential to broaden the applicability of rule-based ASPIC+. However, as is well known from related fields, grounding can be a major computational bottleneck: naive grounding strategies may cause exponential blow-up by generating numerous irrelevant propositional rules (see e.g. (Besin, Hecher, and Woltran 2023)). In ASP, this problem has been extensively addressed, resulting in efficient grounders such as `Gringo` (Gebser et al. 2015) and `i-DLV` (Calimeri et al. 2017). In contrast, no comparable solutions exist specifically for rule-based argumentation.

In this work, we address this gap by proposing a grounding approach for ASPIC+ that leverages engines for Datalog—a declarative language widely used for database querying and reasoning, which can also be seen as the fragment of ASP that excludes negation-as-failure (see e.g. (East and Truszczynski 2000)). Specifically, our contributions are:

1. We define a first-order syntax for rule-based argumentation within the ASPIC+ framework.
2. We propose an intelligent grounding procedure for first-order rule-based ASPIC+ that minimizes the grounding size while preserving the outcomes of the reasoning process. Our approach builds upon query answering in the context of Datalog, albeit incorporating optimizations specific to ASPIC+.
3. We present a prototype grounder, `ANGRY`, which utilizes the Datalog engine `Nemo` (Ivliev et al. 2024).
4. We demonstrate the feasibility of our approach through empirical evaluation in three distinct scenarios. In particular, we evaluate `ANGRY` as part of a ground+solve pipeline for ASPIC+, using the system `ASPforASPIC` (Lehtonen, Wallner, and Järvisalo 2020), and compare it to `Arg2P` (Calegari et al. 2022), the only existing system supporting a first-order ASPIC+-like syntax. To assess its efficiency in the context of ASP, we also compare `ANGRY`—used as an ASP grounder—against the state-of-the-art ASP grounder `Gringo`.

Related Work. The main system we are aware of that supports a first-order ASPIC+-like syntax is `Arg2P` (Calegari et al. 2022), implemented in TUProlog (Ciatto, Calegari, and Omicini 2021). However, it has been shown to be impractical for large instances (Robaldo et al. 2024). Another notable logic programming-inspired argumentation formalism that supports first-order variables is DeLP (García and Simari 2004). However, DeLP employs a different semantics (García, Prakken, and Simari 2020), and to our knowledge, no recent systematic empirical evaluations of DeLP systems exist. Regarding ASPIC+, and similarly for ABA, existing computational work primarily focuses on propositional instances (e.g. also (Craven and Toni 2016; Diller, Gaggl, and Gorczyca 2021; Lehtonen, Wallner, and Järvisalo 2021; Popescu and Wallner 2023; Lehtonen et al. 2023; Lehtonen et al. 2024b) for ABA).

Existing translations between ABA and ASPIC+ (Heyninck 2019), as well as between ASP and ABA (Caminada and Schulz 2017), suggest an alternative approach to grounding ASPIC+ frameworks. This would involve: (1) translating an ASPIC+ framework into an ASP program, (2) applying an ASP grounder, and (3) translating the resulting grounded ASP program back into an ASPIC+ theory. However, this translation-based method faces several limitations. Regarding step (1): existing translations assume propositional ASPIC+. Regarding step (2): ASP grounders are tailored for the stable model semantics, whereas ASPIC+ reasoning may require alternative semantics. For instance, the translations in (Caminada and Schulz 2017) utilize not only stable but also 3-valued stable, well-founded, regular, and ideal semantics. These ASP grounders (e.g., `Gringo`) often include hard-coded optimizations for stable models, which are not easily configurable or extensible. Regarding step (3): ASP grounders are typically designed with the aim of ultimately producing sets of ground atoms (answer sets, via an ASP solver), while ASPIC+ often requires sets of ground arguments (extensions). Optimizations targeted at the former may hinder the latter—as we show in our study. Perhaps most importantly, even if translation-based grounding approaches for ASPIC+ could be developed, our method offers a simpler, more transparent, and modular alternative.

The main goal of this work is to lay the theoretical foundation for grounding ASPIC+, and to demonstrate its practicality through our prototype grounder, `ANGRY`. The system is built on top of the Datalog engine `Nemo` (Ivliev et al. 2024), though in principle, any of the several systems supporting Datalog on offer could be used (see e.g. also (Nenov et al. 2015; Jordan, Scholz, and Subotic 2016; Urbani, Jacobs, and Krötzsch 2016)), including also ASP grounders like `Gringo` and `i-DLV`. This underlines our motivation for choosing Datalog as a foundation: it offers a simpler and more focused computational model than full ASP, and our method does not require the additional features ASP provides.

At the same time, several of the techniques we propose for optimizing ASPIC+ grounding via Datalog are inspired by established ASP grounding techniques. As previously mentioned, since rule-based ASPIC+ subsumes ASP under stable semantics, our grounding procedure also offers a viable alternative for grounding ASP programs via translation to ASPIC+. However, this remains a secondary benefit. Our primary objective is to enable effective grounding for rule-based ASPIC+. Finally, we note that the ASP literature also explores alternative grounding mechanisms as those that interleave grounding and solving (e.g. (Weinzierl, Taupe, and Friedrich 2020)), which fall outside the scope of this work.

2 Background

2.1 Propositional Rule-Based Argumentation in the ASPIC+ Framework

As we indicated in the introduction, among the various instantiations of ASPIC+, the most widely used is arguably that capturing logic programming-style argumentation. We call it *concrete* rule-based ASPIC+ or simply rule-based ASPIC+ for short (see also (Dung and Thang 2014)). Based on (Modgil and Prakken 2018) we start by defining a convenient syntax for the propositional variant of this instance. We assume as given a countably infinite set of propositional atoms, denoted by $\mathcal{L}_0$.

Definition 1. *An* argumentation theory *is a tuple* $\mathcal{T} = (\overline{\ }, \mathcal{R}_s, \mathcal{R}_d, \mathcal{K}_n, \mathcal{K}_p)$. $\overline{\ }$ *is a finite set of expressions* $\overline{s} = \{s_1, \ldots, s_l\}$ *mapping an* $s \in \mathcal{L}_0$ *to its set of* contraries $\{s_1, \ldots, s_l\} \subseteq \mathcal{L}_0$. $\mathcal{R}_s$ *is a finite set of* strict *rules, having the form* $B \to h$ *with* $B \cup \{h\} \subseteq \mathcal{L}_0$. $\mathcal{R}_d$ *is a finite set of* defeasible *rules, which have the form* $n : B \Rightarrow h$ *with* $B \cup \{n, h\} \subseteq \mathcal{L}_0$. *Finally,* $\mathcal{K}_n \subseteq \mathcal{L}_0$ *and* $\mathcal{K}_p \subseteq \mathcal{L}_0$*, for which* $\mathcal{K}_n \cap \mathcal{K}_p = \emptyset$*, are the set of* facts *and* assumptions

respectively[1]*. Both $\mathcal{K}_n$ and $\mathcal{K}_p$ are also finite. The set of rules of $\mathcal{T}$ is then $\mathcal{R} = \mathcal{R}_s \cup \mathcal{R}_d$ and the knowledge base $\mathcal{K} = \mathcal{K}_n \cup \mathcal{K}_p$.*

By minor abuse of notation, we will treat $\overline{\ }$ both as a set of expressions as well as the obvious (partial) function $\mathcal{L}_0 \mapsto 2^{\mathcal{L}_0}$ it induces. For simplicity we will also often represent rules $r = n : B \Rightarrow h \in \mathcal{R}_d$ as $r' = B \Rightarrow h \in \mathcal{R}_d$, with the name n of a defeasible rule (often implicitly) associated to r' via a naming function $N : \mathcal{R}_d \mapsto \mathcal{L}_0$, i.e. $N(r') = n$. Then, in particular, we can use $r = B \rightsquigarrow h$ to denote an arbitrary rule $r \in \mathcal{R}$. We will often also omit the curly braces for sets and rather list their elements, e.g. $r = b_1, \ldots, b_m \rightsquigarrow h$ for the rule above with $B = \{b_1, \ldots, b_m\}$. Arguments in ASPIC+ involve deriving claims from facts and assumptions via the strict and defeasible rules:

Definition 2. *The set of* arguments *of an argumentation theory $\mathcal{T} = (\overline{\ }, \mathcal{R}_s, \mathcal{R}_d, \mathcal{K}_n, \mathcal{K}_p)$ is defined inductively as follows:* **i** *) if $s \in \mathcal{K}_n \cup \mathcal{K}_p$, then $a = s$ is an argument with* conclusion $Conc(a) = s$, premisses $Prem(a) = s$, *and* rules $Rules(a) = \emptyset$; **ii**) *if $a_1, \ldots, a_m$ are arguments and $r = Conc(a_1), \ldots, Conc(a_m) \rightsquigarrow h \in \mathcal{R}$, then $a = a_1, \ldots, a_m \rightsquigarrow h$ is an argument with $Conc(a) = h$, $Prem(a) = \bigcup_{1 \leq i \leq m} Prem(a_i)$, $Rules(a) = \bigcup_{1 \leq i \leq m} Rules(a_i) \cup \{r\}$, and* top-rule $TopRule(a) = r$. *There are no other arguments than those defined by i) and ii).*

Note that by definition arguments are always finite, i.e. they are constructed by finite application of rules. In this work we will often also extend notation to sets in the obvious manner and without explicit definition; then, e.g. for a set of arguments A, $Conc(A) = \bigcup_{a \in A} Conc(a)$. Conflicts between arguments are captured by the notion of attack:

Definition 3. *Let $\mathcal{T} = (\overline{\ }, \mathcal{R}_s, \mathcal{R}_d, \mathcal{K}_n, \mathcal{K}_p)$ be an argumentation theory. An argument a of $\mathcal{T}$* attacks *an argument a' of $\mathcal{T}$ iff a undercuts, rebuts, or undermines a' where* **i)** *a* undercuts *a' (on r) iff $Conc(a) \in \overline{N(r)}$ for a $r \in Rules(a') \cap \mathcal{R}_d$,* **ii)** *$a$* rebuts *$a'$ (on $B \Rightarrow h$) iff $Conc(a) \in \overline{h}$ for a $B \Rightarrow h \in Rules(a') \cap \mathcal{R}_d$, and* **iii)** *$a$* undermines *$a'$ (on s) iff $Conc(a) \in \overline{s}$ for a $s \in Prem(a) \cap \mathcal{K}_p$.*

Note that attacks are always on some *defeasible element* from $Def(a') = (Prem(a') \cap \mathcal{K}_p) \cup (Rules(a') \cap \mathcal{R}_d)$. On the other hand, if we define $Def(r) = \{n, h\}$ and $\overline{r} = \overline{Def(r)} = \overline{n} \cup \overline{h}$ for a $r = n : B \Rightarrow h \in \mathcal{R}_d$, while $\overline{r} = \overline{Def(r')} = Def(r') = \emptyset$ for $r' \in \mathcal{R}_s$, then attacks are always through some element in $\overline{Def(a')} = \bigcup_{u \in Def(a')} \overline{u}$.

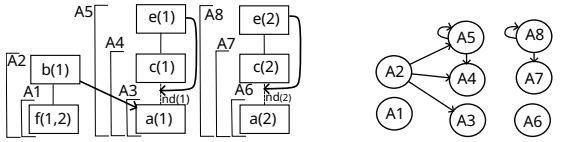
To evaluate ASPIC+ theories, these are traditionally translated into argumentation graphs (Dung 1995):

Definition 4. *An (abstract)* argumentation framework *(or* graph*) (AF) is tuple $(\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is a set of (abstract) arguments and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ the attack relation.*

Definition 5. *For an AF $(\mathcal{V}, \mathcal{E})$, a set $V \subseteq \mathcal{V}$ is* **i)** conflict-free *iff there are no $v, v' \in V$ s.t. $(v, v') \in \mathcal{E}$,* **ii)** admissible *(shorthand: adm) iff V is conflict free and every $v \in V$ is defended by V, where V* defends *v if for every $(v', v) \in \mathcal{E}$, V also* attacks *v', i.e. there is a $v'' \in V$ s.t. $(v'', v') \in \mathcal{E}$,* **iii)** complete *($com$) iff V is admissible and includes every $v \in \mathcal{V}$ it defends,* **iv)** grounded *(grd) iff V is subset-minimal among the complete sets,* **v)** preferred *(prf) iff V is subset-maximal among the complete sets,* **vi)** stable *(stb) iff V is admissible and attacks every $v \in \mathcal{V} \setminus V$.*

Definition 6. *The* AF defined by the argumentation theory $\mathcal{T}$ *is the argumentation graph $(Args(\mathcal{T}), Atts(\mathcal{T}))$ where $Args(\mathcal{T})$ are all the arguments of $\mathcal{T}$ and $Atts(\mathcal{T})$ is the attack relation among arguments induced by $\mathcal{T}$. $S \subseteq \mathcal{L}_0$ is credulously (skeptically)* acceptable *for $\theta \in \{adm, com, grd, prf, stb\}$ iff there exists a θ-extension*[2] *(for all θ-extensions) $A \subseteq Args(\mathcal{T})$, $S \subseteq Conc(A)$.*

Example 1. *Consider the argumentation theory $\mathcal{T} = (\overline{\ }, \mathcal{R}_s, \mathcal{R}_d, \mathcal{K}_n, \mathcal{K}_p)$ with $\mathcal{K}_n = \{f(1,2)\}$*[3]*, $\mathcal{K}_p = \{a(1), a(2)\}$, $\mathcal{R}_s = \{f(1,2) \rightarrow b(1), c(1) \rightarrow e(1), c(2) \rightarrow e(2)\}$, $\mathcal{R}_d = \{n_d(1) : a(1) \Rightarrow c(1), n_d(2) : a(2) \Rightarrow c(2)\}$, and $\overline{\ } = \{\overline{a(1)} = b(1), \overline{a(2)} = b(2), \overline{c(1)} = d(1), \overline{c(2)} = d(2), \overline{n_d(1)} = e(1), \overline{n_d(2)} = e(2)\}$. The set of arguments (A1-A8) of $\mathcal{T}$ and the attacks (arrows) between them is shown in Fig. 1a. The* AF *defined by $\mathcal{T}$ is depicted in Fig. 1b. For this framework, the unique complete extension is $\{A1, A2, A6\}$, while the admissible sets are all of the subsets of $\{A1, A2, A6\}$. This means that the three arguments ($A1$, $A2$, $A6$) are all credulously accepted under the admissible semantics, whereas they are skeptically accepted under the complete semantics. In terms of claims, this implies that the conclusions $(f(1,2), b(1), a(2))$ of the three arguments are all credulously (resp. skeptically) accepted under the admissible (resp. complete) semantics. Since the complete extension is unique in this case, the* grounded *and* preferred *extensions coincide with the complete extension. In contrast, there is no stable extension in this framework.*



(a) Argumentation theory (b) Argumentation graph

Figure 1: Argumentation theory and induced AF from Example 1.

2.2 Datalog

To define Datalog (see, for instance, (Abiteboul, Hull, and Vianu 1995)) we extend the language of atoms $\mathcal{L}_0$ to a language $\mathcal{L}$ to be built from mutually disjoint, countably infinite sets of *constants* C, *variables* V, and *predicates* P. A *term* t is an element $t \in \mathrm{C} \cup \mathrm{V}$. We represent a list of terms

[1]These are called axiom and ordinary premisses in the ASPIC+ framework; we use the notions "facts" and "assumptions" to denote the concrete case in which they are propositional atoms.

[2]An *extension* is a set of arguments that satisfies the criteria of the semantics.

[3]In this section we treat an expression like $f(1,2)$ as a propositional atom; later we define how such atoms are obtained via grounding of first-order atoms.

$t_1, \ldots, t_m$ as $\vec{t}$, and we treat such lists as sets, if appropriate. An (extended) *atom* is then an expression $p(\vec{t})$ where $p \in \mathrm{P}$. An atom is *ground* if no variables occur in the atom (i.e. all terms are constants). For an expression ϕ (usually an atom or set of atoms), we denote by $\phi(\vec{X})$ that ϕ uses (exactly) the variables $\vec{X}$. Throughout this work we will use upper-case letters for variables and lower-case letters for constants.

Definition 7. *A* Datalog program *is a finite set of* (Datalog) rules *of the form $B(\vec{X}) \to h(\vec{Y})$ where $h(\vec{Y}) \in \mathcal{L}$ and $B(\vec{X}) \subseteq \mathcal{L}$. The rule is* safe *iff $\vec{Y} \subseteq \vec{X}$.*

Datalog programs are evaluated by grounding the rules and then computing their consequences:

Definition 8. *For a Datalog program $\mathcal{P}$, the* Herbrand universe *$U_\mathcal{P}$ is the set of all constants occurring in $\mathcal{P}$. The* Herbrand literal base *$B_\mathcal{P}$ is the set of all ground atoms $p(\vec{t})$ with p occurring in $\mathcal{P}$ and $\vec{t} \subseteq U_\mathcal{P}$. The* grounding of a rule $r = B(\vec{X}) \to h(\vec{Y}) \in \mathcal{P}$ *is defined as $gr(r, \mathcal{P}) = \{r\sigma \mid \sigma : \vec{X} \cup \vec{Y} \mapsto U_\mathcal{P}\}$. The* grounding of the program $\mathcal{P}$, *$gr(\mathcal{P})$, is the union of the grounding of all of its rules.*

Definition 9. *The* immediate consequence operator *$T_\mathcal{P}$ for a Datalog program $\mathcal{P}$ and a set of ground atoms $I \subseteq B_\mathcal{P}$ is $T_\mathcal{P}(I) = \{h \mid B \to h \in gr(\mathcal{P}), B \subseteq I\}$. The least fixed-point of $T_\mathcal{P}$ is $T^\infty_\mathcal{P} = \bigcup_{i \geq 0} T^i_\mathcal{P}$, where $T^0_\mathcal{P} = \emptyset$ and $T^{i+1}_\mathcal{P} = T_\mathcal{P}(T^i_\mathcal{P})$. A ground atom $p(\vec{t})$ is* derived *from $\mathcal{P}$ iff $p(\vec{t}) \in T^\infty_\mathcal{P}$. A* query *$\langle \mathcal{P}, q \rangle$ asks for all atoms $q(\vec{t})$ that can be derived from $\mathcal{P}$.*

Datalog can be extended to include a limited form of negation:

Definition 10. *Datalog with* stratified negation *extends Datalog by allowing negated atoms $\sim p(\vec{t})$ in rule bodies of a Datalog program $\mathcal{P}$ as long as $\mathcal{P}$ has a* stratification*. The latter is a function l that assigns each predicate occurring in $\mathcal{P}$ a natural number such that for every rule $r \in \mathcal{P}$ with $p \in \mathrm{P}$ occurring in the head of r, if $p' \in \mathrm{P}$ occurs in the body of r, then $l(p) \geq l(p')$, while if $p' \in \mathrm{P}$ occurs in a negated atom in the body of r, then $l(p) > l(p')$.*

Stratified Datalog programs are partitioned into strata based on predicate dependencies, with each stratum containing rules defining predicates at the same level. Evaluation proceeds bottom-up: at each stratum, rules are applied using facts derived so far—negated atoms are evaluated as true (and, thus, can be essentially deleted from rules) if no matching positive fact is derived in the prior lower stratum.

3 Grounding Rule-Based Argumentation

3.1 First-Order Rule-Based Argumentation in the ASPIC+ Framework

Although a first-order (F.O.) syntax for rule-based ASPIC+ is often used, it has, to our knowledge, not been formally defined. We provide one following common definitions of logic-programming (see e.g. (Faber 2020)). For this we again make use of a language $\mathcal{L}$ of atoms as in Section 2.2, while $\mathcal{L}_0 \subseteq \mathcal{L}$ denotes the ground atoms.

Definition 11. *A* (first-order) argumentation theory *is a tuple $\mathcal{T} = (\overline{\ }, \mathcal{R}_s, \mathcal{R}_d, \mathcal{K}_n, \mathcal{K}_p)$. $\overline{\ }$ is a finite set of (safe)* contrary expressions *$\overline{s(\vec{X})} = S(\vec{X})$ with $s(\vec{X}) \in \mathcal{L}$, $S(\vec{X}) \subseteq \mathcal{L}$. Moreover, no constants occur in either $s(\vec{X})$ or $S(\vec{X})$. $\mathcal{R}_s$ is a finite set of (safe)* strict *rules, having the form $B(\vec{X}) \to h(\vec{Y})$ with $h(\vec{Y}) \in \mathcal{L}$, $B(\vec{X}) \subseteq \mathcal{L}$, $\vec{Y} \subseteq \vec{X}$. $\mathcal{R}_d$ is a finite set of (safe)* defeasible *rules, which have the form $n(\vec{Z}) : B(\vec{X}) \Rightarrow h(\vec{Y})$ with $n(\vec{Z}) \in \mathcal{L}$, $h(\vec{Y}) \in \mathcal{L}$, $B(\vec{X}) \subseteq \mathcal{L}$, $\vec{Y} \subseteq \vec{X}$, $\vec{Z} \subseteq \vec{X}$. Finally, $\mathcal{K}_n \subseteq \mathcal{L}_0$ and $\mathcal{K}_p \subseteq \mathcal{L}_0$, for which $\mathcal{K}_n \cap \mathcal{K}_p = \emptyset$, are the set of* facts *and* assumptions *respectively. Both $\mathcal{K}_n$ and $\mathcal{K}_p$ are also finite.*

Example 2. *An example of a first-order argumentation theory is $\mathcal{T} = (\overline{\ }, \mathcal{R}_s, \mathcal{R}_d, \mathcal{K}_n, \mathcal{K}_p)$ where $\overline{\ } = \{\overline{a(X)} = b(X), \overline{n_d(X)} = e(X), \overline{c(X)} = d(X)\}$, $\mathcal{R}_s = \{f(X,Y) \to b(X), c(X) \to e(X)\}$, $\mathcal{R}_d = \{n_d(X) : a(X) \Rightarrow c(X)\}$, $\mathcal{K}_p = \{a(1), a(2)\}$, $\mathcal{K}_n = \{f(1,2)\}$. As will be shown in the following sections, the grounding of this first-order theory is equivalent, in terms of induced AFs, to the propositional theory of Example 1 depicted in Fig 1a.*

As to the restrictions in Definition 11, safety of rules is a common restriction from logic-programming to ensure termination when solving. The restriction of facts and assumptions to be ground is to distinguish these from strict rules and defeasible rules (i.e. if non-ground assumptions, for instance, are needed, they can be defined via defeasible rules). As to the restriction for contrary expressions, this is for simplicity; if a contrary expression like $\overline{p(X)} = q(X,Y,c)$ is needed, it can be defined via the contrary expression $\overline{p(X)} = q'(X)$ and the rule $q(X,Y,c) \to q'(X)$.

To obtain a ground (i.e. propositional) theory from a F.O. argumentation theory the same approach as for Datalog (Section 2.2) can be used. I.e. for a theory $\mathcal{T}$, the Herbrand universe $U_\mathcal{T}$ is the set of all constants occurring in $\mathcal{T}$. Then, the grounding of $\mathcal{T}$, $gr(\mathcal{T})$, is obtained by grounding each of the contrary expressions and rules in $\mathcal{T}$, i.e. replacing all variables for elements of $U_\mathcal{T}$ in all possible ways. The semantics of $\mathcal{T}$ is then obtained by evaluating $gr(\mathcal{T})$ via the induced AF as explained in Section 2.1.

3.2 Grounding via Datalog: The Basics

Naively grounding argumentation theories as described at the end of Section 3.1 will often produce rules that do not form part of any argument.

Example 3. *Consider the argumentation theory $\mathcal{T}$ of Example 2 and let us focus on the strict rule $f(X,Y) \to b(X)$. A naive grounding produces $gr(\mathcal{R}_s) = \{f(1,1) \to b(1), f(1,2) \to b(1), f(2,1) \to b(2), f(2,2) \to b(2)\}$. The only arguments derived by these rules and included in the AF for $\mathcal{T}$ use $f(1,2)$ as a premise and $b(1)$ as a conclusion, via the rule $f(1,2) \to b(1)$ (i.e. the arguments A1 and A2 depicted in Fig 1a). The same set of arguments would be produced with a more efficient grounding that only produces the rule $f(1,2) \to b(1)$.*

We thus now introduce a transformation of an ASPIC+ theory $\mathcal{T}$ that generates a set of Datalog rules, which helps

to produce a smaller grounding that excludes unnecessary rules, while preserving the extensions of $\mathcal{T}$.

Transformation 1. *Let $\mathcal{T} = (\bar{}, \mathcal{R}_s, \mathcal{R}_d, \mathcal{K}_n, \mathcal{K}_p)$ be an argumentation theory. For a rule $r = B(\vec{X}) \rightarrow h(\vec{Y}) \in \mathcal{R}_s$ or $r = n(\vec{Z}) : B(\vec{X}) \Rightarrow h(\vec{Y}) \in \mathcal{R}_d$ we denote the transformation into Datalog rules by $\hat{r}$ as the set of rules consisting of the following:*

$$B(\vec{X}) \rightarrow n_r(\vec{X}) \quad (1)$$

$$n_r(\vec{X}) \rightarrow h(\vec{Y}) \quad (2)$$

where n_r is a fresh predicate that is unique for the rule. For the defeasible rule $r \in \mathcal{R}_d$ we also need the following additional Datalog rule:

$$n_r(\vec{X}) \rightarrow n(\vec{Z}) \quad (3)$$

Facts and assumptions are transformed into Datalog rules with empty bodies. In particular, for any ground atom $b \in \mathcal{K}_n \cup \mathcal{K}_p$ we define $\hat{b} =\!\!\rightarrow b$. Then, the transformation of $\mathcal{T}$ to the respective Datalog program is $\mathcal{P}_\mathcal{T} = \{\hat{\mathcal{R}}_s \cup \hat{\mathcal{R}}_d \cup \hat{\mathcal{K}}_n \cup \hat{\mathcal{K}}_p\}$.

Once we have created the Datalog program $\mathcal{P}_\mathcal{T}$ by applying Transformation 1 for a theory $\mathcal{T}$, we ground $\mathcal{T}$ by querying a Datalog engine for each strict and defeasible rule $r \in \mathcal{R}$ with the respective query $(\mathcal{P}_\mathcal{T}, n_r)$ for the introduced auxiliary predicates. For all obtained ground atoms $n_r(\vec{a}_i)$, with $0 \leq i \leq l$, we make the ground substitutions $\{r\sigma_i \mid \sigma_i : \vec{X} \mapsto \vec{a}_i\}$ as described in Algorithm 1. The union over all ground substitutions of a rule r for the Datalog program $\mathcal{P}_\mathcal{T}$ is defined as $gr_{DL}(r, \mathcal{P}_\mathcal{T}) = \bigcup_{i=0}^{l} r\sigma_i$.

Algorithm 1 Grounding of an ASPIC+ rule

Input: A theory $\mathcal{T}$, $r \in \mathcal{R}$, $\mathcal{P}_\mathcal{T}$ with $n_r(\vec{X}) \in \mathcal{P}_\mathcal{T}$
Output: $gr_{DL}(r, \mathcal{P}_\mathcal{T})$

```
gr_DL(r, P_T) ⟵ ∅
for all 0 ≤ i ≤ l answers n_r(a_i) to ⟨P_T, n_r⟩ do
    // each n_r(a_i) gives rise to a ground instance of r
    {σ_i : X_k ↦ a_k | 1 ≤ k ≤ |X|}
    gr_DL(r, P_T) ⟵ gr_DL(r, P) ∪ {rσ_i}
end for
return gr_DL(r, P_T)
```

Grounding of a contrary expression $c : \overline{s(\vec{X})} = S(\vec{X})$ is obtained in an analogous way (thus, we will often say that we also ground a contrary expression following Algorithm 1). We query the Datalog engine with the query $(\mathcal{P}_\mathcal{T}, s)$ and for each answer $s(\vec{a}_i)$ with $0 \leq i \leq l$ we make the substitution $\{c\sigma_i \mid \sigma_i : X_k \mapsto a_k \mid 1 \leq k \leq |\vec{X}|\}$. Then, the union over all ground substitutions of a contrary expression c for the Datalog program $\mathcal{P}_\mathcal{T}$ is defined as $gr_{DL}(c, \mathcal{P}_\mathcal{T}) = \bigcup_{i=0}^{l} c\sigma_i$. For $\mathcal{T} = (\bar{}, \mathcal{R}_s, \mathcal{R}_d, \mathcal{K}_n, \mathcal{K}_p)$ we thus obtain the grounding $gr_{DL}(\mathcal{T}) = (gr_{DL}(\bar{}), gr_{DL}(\mathcal{R}_s), gr_{DL}(\mathcal{R}_d), \mathcal{K}_n, \mathcal{K}_p)$, with $gr_{DL}(\bar{}) = \bigcup_{c \in} - gr_{DL}(c, \mathcal{P}_\mathcal{T})$ and analogously for $gr_{DL}(\mathcal{R}_s)$ and $gr_{DL}(\mathcal{R}_d)$.

Example 4. *In this example we show how a propositional argumentation theory $gr_{DL}(\mathcal{T})$ is obtained by grounding the F.O. theory from Example 2 using the Datalog program $\mathcal{P}_\mathcal{T}$ and Algorithm 1. To obtain $\mathcal{P}_\mathcal{T}$, we first transform $f(1,2) \in \mathcal{K}_n$, $a(1) \in \mathcal{K}_p$ and $a(2) \in \mathcal{K}_p$ into the Datalog rules $\rightarrow f(1,2)$, $\rightarrow a(1)$ and $\rightarrow a(2)$, and add them to $\mathcal{P}_\mathcal{T}$. Next, we apply Transformation 1 to strict and defeasible rules. For instance, $f(X,Y) \rightarrow b(X)$ results in the Datalog rules $f(X,Y) \rightarrow n_r(X,Y)$ and $n_r(X,Y) \rightarrow b(X)$, both of which are added to $\mathcal{P}_\mathcal{T}$. Similarly, the defeasible rule $n_d(X) : a(X) \Rightarrow c(X)$ results in $a(X) \rightarrow n'_r(X)$ and $n'_r(X) \rightarrow c(X)$, along with the additional rule $n'_r(X) \rightarrow n_d(X)$. Now, we ground $\mathcal{T}$ by invoking Algorithm 1 for each rule of the theory. For example, consider the rule $f(X,Y) \rightarrow b(X)$. First, we execute the query $(\mathcal{P}_\mathcal{T}, n_r)$, producing the ground atom $n_r(1,2)$ and the substitution $\{\sigma : X \mapsto 1, Y \mapsto 2\}$. Then, we apply this substitution to the rule obtaining $f(1,2) \rightarrow b(1)$. By applying the same procedure to the other rules ($c(X) \rightarrow e(X)$ and $n_d(X) : a(X) \Rightarrow c(X)$) we obtain: $c(1) \rightarrow e(1)$; $c(2) \rightarrow e(2)$; $n_d(1) : a(1) \Rightarrow c(1)$ and $n_d(2) : a(2) \Rightarrow c(2)$. Now, let us consider the contrary relation $\overline{a(X)} = b(X)$. We execute the query $(\mathcal{P}_\mathcal{T}, a)$ obtaining the substitutions $\{\sigma_1 : X \mapsto 1\}$ and $\{\sigma_2 : X \mapsto 2\}$, resulting in the propositional contrary relations $\overline{a(1)} = b(1)$ and $\overline{a(2)} = b(2)$. The obtained propositional rules and contrary relations, together with the initial facts and assumptions ($\mathcal{K}_n$ and $\mathcal{K}_p$), forms the grounded argumentation theory $gr_{DL}(\mathcal{T})$, which, as anticipated, coincides with the propositional argumentation theory in Example 1.*

Lemma 1 expresses that the AFs defined by $gr(\mathcal{T})$ and $gr_{DL}(\mathcal{T})$ are the same, from which it clearly follows (Theorem 1), that $\mathcal{T}$ and $gr_{DL}(\mathcal{T})$ have the same extensions.

Lemma 1 ($\star$[4]). *Let $\mathcal{T}$ be an argumentation theory and $gr_{DL}(\mathcal{T})$ the grounding via the Datalog program $\mathcal{P}_\mathcal{T}$ as per Transformation 1. Then, $Args(gr(\mathcal{T})) = Args(gr_{DL}(\mathcal{T}))$ and $Atts(gr(\mathcal{T})) = Atts(gr_{DL}(\mathcal{T}))$.*

Theorem 1 ($\star$)**.** *Let $\mathcal{T}$ be an argumentation theory and $gr_{DL}(\mathcal{T})$ the grounding via the Datalog program $\mathcal{P}_\mathcal{T}$ as per Transformation 1. Then, $\theta(\mathcal{T}) = \theta(gr(\mathcal{T})) = \theta(gr_{DL}(\mathcal{T}))$ for $\theta \in \{adm, com, grd, prf, stb\}$.*

3.3 Improvements

Non-Approximated Predicates. In Transformation 1 we ignore the distinction between defeasible and non-defeasible elements (assumptions and defeasible rules on the one side, facts and strict rules on the other). To obtain all possible arguments of an argumentation theory this is fine. But we can further simplify the grounding by generating rules that will be used only in acceptable arguments (i.e. those included in some extension).

Example 5. *Consider the propositional theory obtained in Example 4 (via grounding the F.O. theory from Example 2 via the procedure described in Section 3.2) and depicted in Figure 1a. Any argument using $a(1)$ as a premise (namely*

[4] See arXiv version of this paper for proofs.

$A3$, $A4$, and $A5$) is undermined by $A2$, which is itself not attacked by any argument. As a result, neither $A3$, $A4$ nor $A5$ will be included in any extension for any of the semantics we consider in this work. Hence, the assumption and rules that generate such arguments (i.e. $a(1)$, $n_d(1) : a(1) \Rightarrow c(1)$, and $c(1) \rightarrow e(1)$) can also be excluded from the grounding, further reducing the size of the grounding while preserving the extensions.

We introduce the notion of approximated and non-approximated predicates to help us formalise when rules and assumptions can be removed from the grounding in Definition 12. The intuition behind the distinction between these different predicates is as follows: non-approximated predicates are those for which grounding can fully determine derivability of the ground instances within some extension. Approximated predicates leave determination of this to the solver.

Definition 12. *Let $\mathcal{T} = (\bar{\ }, \mathcal{R}_s, \mathcal{R}_d, \mathcal{K}_n, \mathcal{K}_p)$ be an argumentation theory and let p first be a predicate and $r \in \mathcal{R}$ a rule whose head is an atom containing p. Then, predicate p* depends positively *on a predicate p' if the body of r contains an atom containing p'. The predicate p* depends negatively *on p' if p' appears in the contrary expression of one of the defeasible elements of r ($\overline{Def(r)}$). Let now p be a predicate occurring in an assumption atom l ($l \in \mathcal{K}_p$). Then, p* depends negatively *on p' if p' occurs in the contrary expression of l. The set of* approximated *predicates is the minimal set containing a predicate p if one of the following cases holds: i) p depends on an approximated predicate, or ii) there is a circular sequence of dependencies $p = p_1, p_2, \ldots, p_n = p$, where each p_{i+1} depends on p_i and there is a p_{i+1} that depends negatively on p_i. Any predicate that is not approximated we call* non-approximated.

Example 6. *Consider the argumentation theory of Example 2. To compute the approximated and non-approximated predicates of the theory, we analyze the positive and negative dependencies between predicates. For example, consider the rules $r_1 = n_d(X) : a(X) \Rightarrow c(X)$ and $r_2 = c(X) \rightarrow e(X)$, and the contrary expression $\overline{n_d(X)} = e(X)$. We consider the predicates appearing in the heads of the rules, namely c and e. First, from the rule r_2, we see that e depends positively on c. Next, rule r_1 shows that c depends negatively on e. This is because e appears in the contrary expression of $n_d(X)$, which is a defeasible element of r_1. Therefore, there exist circular sequences of dependencies with at least one negative relation: e, c, e and c, e, c. This indicates that both e and c are approximated predicates. In contrast, all other predicates in the theory are non-approximated predicates.*

Using the knowledge of which predicates are non-approximated, we can simplify the grounding:

Transformation 2. *Let $\mathcal{T} = (\bar{\ }, \mathcal{R}_s, \mathcal{R}_d, \mathcal{K}_n, \mathcal{K}_p)$ be an argumentation theory. A rule $r = B(\vec{X}) \rightarrow h(\vec{Y}) \in \mathcal{R}_s$ or $r = n(\vec{X}) : B(\vec{Y}) \Rightarrow h(\vec{Z}) \in \mathcal{R}_d$, given $\{l_1(\vec{X_1}), \ldots, l_n(\vec{X_n})\} \subseteq \overline{Def(r)}$ where the l_i are exactly the non-approximated predicates in $\overline{Def(r)}$, gives rise to the set $\hat{r}$ consisting of the following Datalog rules:*

$$B(\vec{X}), \sim l_1(\vec{X_1}), \ldots, \sim l_n(\vec{X_n}) \rightarrow n_r(\vec{X}) \quad (1)$$

$$n_r(\vec{X}) \rightarrow h(\vec{Y}) \quad (2)$$

where n_r is a fresh predicate that is unique for the rule. For the defeasible rule $r \in \mathcal{R}_d$ the following rule also is part of $\hat{r}$ as in Transformation 1:

$$n_r(\vec{X}) \rightarrow n(\vec{Z}) \quad (3)$$

Facts are also transformed as in Transformation 1: for any $b \in \mathcal{K}_n$, $\hat{b} =\!\!\rightarrow b$. Assumptions are transformed analogously to defeasible rules. I.e. for a ground atom $b \in \mathcal{K}_p$ with $\{l_1(\vec{t_1})), \ldots, l_n(\vec{t_n})\} \subseteq \bar{b}$ where l_i are exactly the non-approximated predicates appearing in $\bar{b}$, we define $\hat{b} = \sim l_1(\vec{t_1}), \ldots, \sim l_n(\vec{t_n}) \rightarrow b$. Then, the transformation of the argumentation theory $\mathcal{T}$ to the respective Datalog program is $\mathcal{P}_\mathcal{T} = \{\hat{\mathcal{R}_s} \cup \hat{\mathcal{R}_d} \cup \hat{\mathcal{K}_n} \cup \hat{\mathcal{K}_p}\}$.

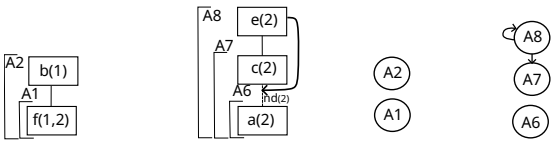
We observe that, because only non-approximated predicates appear negated in bodies of rules of $\mathcal{P}_\mathcal{T}$ as per Transformation 2, $\mathcal{P}_\mathcal{T}$ uses stratified negation as defined in Section 2.2. The grounding of $\mathcal{T}$ via $\mathcal{P}_\mathcal{T}$ as per Transformation 2 is then obtained by grounding the contrary relation and rules of $\mathcal{T}$ by querying $\mathcal{P}_\mathcal{T}$ as in Section 3.2. The difference is that now also assumptions are queried, i.e. $gr_{DL}(\mathcal{K}_p) = \{b(\vec{t}) \in \mathcal{K}_p \cap \langle \mathcal{P}_\mathcal{T}, b \rangle\}$. For $\mathcal{T} = (\bar{\ }, \mathcal{R}_s, \mathcal{R}_d, \mathcal{K}_n, \mathcal{K}_p)$ we thus obtain the grounding $gr_{DL}(\mathcal{T}) = (gr_{DL}(\bar{\ }), gr_{DL}(\mathcal{R}_s), gr_{DL}(\mathcal{R}_d), \mathcal{K}_n, gr_{DL}(\mathcal{K}_p))$[5].

Example 7. *We ground the argumentation theory from Example 2 using the Datalog program $\mathcal{P}_\mathcal{T}$ obtained via Transformation 2. As to the facts and strict rules, Transformation 2 is exactly as Transformation 1, i.e. as in Example 4 (since, in particular, $Def(r) = \emptyset$ for $r \in \mathcal{R}_s$). Thus, we focus on the transformation of defeasible rules and assumptions. Consider the defeasible rule $n_d(X) : a(X) \Rightarrow c(X)$, which includes two defeasible elements ($c(X)$ and $n_d(X)$), whose contraries are $e(X)$ and $d(X)$. As discussed in Example 6, e is an approximated predicate, and therefore it does not appear in the Datalog rules generated by Transformation 2. The resulting rules are $a(X), \sim d(X) \rightarrow n'_r(X)$ and $n'_r(X) \rightarrow c(X)$, along with the additional rule $n'_r(X) \rightarrow n_d(X)$. The assumptions $a(1)$ and $a(2)$ are translated into $\sim b(1) \rightarrow a(1)$ and $\sim b(2) \rightarrow a(2)$ respectively, since the contrary of $a(1)$ (resp. $a(2)$) is $b(1)$ (resp. $b(2)$) and b is a non-approximated predicate.*

Next, we ground $\mathcal{T}$ by invoking Algorithm 1 for each rule of the theory, as already shown in Example 4. Notably, we obtain fewer propositional rules compared with the previous example. For instance, consider again the rule $n_d(X) : a(X) \Rightarrow c(X)$, which we ground by performing the query $(\mathcal{P}_\mathcal{T}, n'_r)$ and obtaining the single substitution $\{\sigma_1 : X \mapsto 1\}$, instead of the two substitutions obtained in Example 4. In fact, the second substitution $\{\sigma_2 : X \mapsto 2\}$ can not be derived due to the negative terms introduced by Transformation 2. The grounding of contrary relations is

[5]We use the same notation, i.e. $\mathcal{P}_\mathcal{T}$ and $gr_{DL}(\mathcal{T})$, for the different versions of the Datalog program and groundings we introduce.

also impacted by the introduced optimization. For example, grounding the contrary relation $\overline{a(X)} = b(X)$ now results in a single propositional expression, $\overline{a(2)} = b(2)$, instead of the two previously derived. On the other hand, $a(1) \notin gr_{DL}(\mathcal{K}_p)$ as $a(1) \notin \langle \mathcal{P}_\mathcal{T}, a\rangle$. Finally, the obtained assumptions, rules and contrary relations, together with the initial facts, forms the grounded argumentation theory $gr_{DL}(\mathcal{T})$, depicted in Fig 2 together with the induced AF. Note that, although the arguments A3, A4, A5 from the theory of Example 1 are lost, the unique complete (as well as grounded and preferred) extension is still $\{A1, A2, A6\}$ (while there is also no stable extension) and, thus, also the acceptable conclusions are the same as in Example 1. In this example the admissible extensions are also the same but we show in Example 8 that this is not guaranteed when grounding via Transformation 2.



(a) Argumentation theory (b) Argumentation graph

Figure 2: Argumentation theory and induced AF from Example 7.

Lemma 2 expresses the relation between the AFs induced by $gr(\mathcal{T})$ and the improved $gr_{DL}(\mathcal{T})$. For this, for an argumentation theory, we define the *certain* arguments to be those that are included in every complete extension of the theory. If, on the other hand, an argument is not attacked by a certain argument we call it *tentative*. For a set of arguments A, we then denote $Args^{!}(A)$ to be the certain arguments in A, while $Args^{?!}(A)$ are the tentative arguments in A. Note, in particular, that certain arguments are also tentative. We also define $Args(A)\downarrow_P$ to be those arguments in A with conclusions making use of predicates in a set of predicates P. Similarly, $Atts(gr(\mathcal{T}))\downarrow_{Args(gr_{DL}(\mathcal{T}))} = \{(a, a') \in Atts(gr(\mathcal{T})) \mid \{a, a'\} \subseteq Args(gr_{DL}(\mathcal{T}))\}$. Then, Lemma 2 indicates that the AFs induced by $gr(\mathcal{T})$ and $gr_{DL}(\mathcal{T})$ coincide on the certain arguments. That these are the arguments that count for all semantics that produce complete extensions is expressed in Theorem 2.

Lemma 2 ($\star$)**.** *Let $\mathcal{T}$ be an argumentation theory and $gr_{DL}(\mathcal{T})$ the grounding via the Datalog program $\mathcal{P}_\mathcal{T}$ as per Transformation 2. Then, $Args^{?!}(gr(\mathcal{T})) \subseteq Args(gr_{DL}(\mathcal{T})) \subseteq Args(gr(\mathcal{T}))$ and $Atts(gr_{DL}(\mathcal{T})) = Atts(gr(\mathcal{T}))\downarrow_{Args(gr_{DL}(\mathcal{T}))}$.*

Theorem 2 ($\star$)**.** *Let $\mathcal{T}$ be an argumentation theory and $gr_{DL}(\mathcal{T})$ the grounding via the Datalog program $\mathcal{P}_\mathcal{T}$ as per Transformation 2. Then, $\theta(\mathcal{T}) = \theta(gr(\mathcal{T})) = \theta(gr_{DL}(\mathcal{T}))$ for $\theta \in \{com, grd, prf, stb\}$.*

Example 8. *Note, on the other hand, that it is not the case that $adm(gr(\mathcal{T})) = adm(gr_{DL}(\mathcal{T}))$. This can be seen by considering the simple (ground) argumentation theory $\mathcal{T}$ with $\mathcal{K}_n = \{a\}$, $\mathcal{K}_p = \{b, c\}$, $\mathcal{R} = \emptyset$, and $\bar{\ } = \{\bar{b} = \{a\}, \bar{c} = \{b\}\}$. Then, $gr(\mathcal{T}) = \mathcal{T}$. On the other hand, $\mathcal{P}_\mathcal{T} = \{\rightarrow a, \sim a \rightarrow b, \sim b \rightarrow c\}$ and, hence, $gr_{DL}(\mathcal{T})$ is formed by $\mathcal{K}'_n = \mathcal{K}_n = \{a\}$, $\mathcal{R}' = \mathcal{R} = \emptyset$, while $\mathcal{K}'_p = \{c\}$ and $\bar{\ }' = \{\bar{c} = \{b\}\}$ (but $b \notin \mathcal{K}'_p$). Then, $adm(gr(\mathcal{T})) = \{\emptyset, \{a\}, \{a, c\}\}$, while $adm(gr_{DL}(\mathcal{T})) = adm(gr(\mathcal{T})) \cup \{\{c\}\}$.*

Facts. In the previous section we simplified the grounding by excluding rules appearing only in arguments that do not appear in any extension. We now look at what is, to some extent, the dual view.

Example 9. *Consider the propositional theory obtained in Example 7 (via grounding the F.O. theory from Example 2 using the procedure described in the previous section) and depicted in Figure 2. Consider the fact $f(1, 2)$ and the strict rule $f(1, 2) \rightarrow b(1)$ of Example 2, which together make up the argument A2. This argument, as it is made up only of facts and strict rules, is immune to attacks, which means that the argument A2 appears in all extensions for all the semantics we consider in this work. This means also that the grounding can be simplified by adding the claim $b(1)$ to the fact base and, as a consequence, removing the strict rule from the set of rules as it becomes redundant.*

Towards simplifying the grounding along the lines of the idea explored in Example 9, note that the dependency relations between predicates of an argumentation theory from Definition 12 can be extended to rules. In particular, for the positive dependency relation: a rule r depends positively on r' if the predicate p appearing in the head of r depends positively on the predicate p' occurring in the head of r' (i.e. p' occurs in the body of r as well as the head of r'). The positive dependency graph $G^+_\mathcal{R}$ for the set of rules $\mathcal{R}$ of an argumentation theory $\mathcal{T}$ is then a directed graph having rules as nodes and an edge from r' to r indicating that r depends positively on r'. Given the dependency graph $G^+_\mathcal{R}$, the strongly connected components of $G^+_\mathcal{R}$ form a topological ordered partition into sub-graphs $L_\mathcal{T} = (C_1, C_2, \ldots, C_n)$. Algorithm 2 then improves on the previous groundings by also (in lines 3-18) iterating over the strongly connected components of $G^+_\mathcal{R}$ and collecting facts, while removing strict rules, as illustrated in Example 9[6].

Example 10. *We ground the argumentation theory from Example 2 using the Datalog program $\mathcal{P}_\mathcal{T}$ obtained via Transformation 2 and Algorithm 2, which incorporates the optimization described above. Since this optimization does not affect assumptions or defeasible rules, we only focus on the strict rules. Consider the rule $f(X, Y) \rightarrow b(X)$. At line 4 of Algorithm 2, we call Algorithm 1, which generates the propositional rule $f(1, 2) \rightarrow b(1)$. Then, at line 9, we remove the fact $f(1, 2)$ from the body of the rule, resulting in the rule $\rightarrow b(1)$. Because this is now a strict rule with an empty body, at lines 11–12 we add $b(1)$ to $gr_{DL}(\mathcal{K}_n)$ and remove $\rightarrow b(1)$ from $gr_{DL}(\mathcal{R}_s)$. The resulting argumentation theory and argumentation graph is depicted in Figure 3. In this case, the unique complete extension is $\{A1, A2, A9\}$, that differs from the one obtained in Example 2. It is worth noting that, although extensions are not*

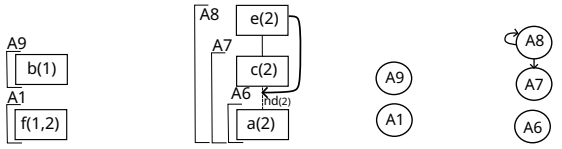
[6] In the algorithm $B(r)$ denotes the body of a rule r, and $h(r)$ the head.

Algorithm 2 Grounding of an ASPIC+ theory

Input: A theory $\mathcal{T} = (^{-}, \mathcal{R}_s, \mathcal{R}_d, \mathcal{K}_n, \mathcal{K}_p)$, $L_{\mathcal{T}}$, the Datalog program $\mathcal{P}_{\mathcal{T}}$ obtained via Transformation 2.
Output: $gr_{DL}(\mathcal{T})$.

$gr_{DL}(\mathcal{K}_n) \leftarrow \mathcal{K}_n$
$gr_{DL}(\mathcal{R}) \leftarrow \emptyset$
for all $C \in L_{\mathcal{T}}$ **do**
  // Ground rules in C via Algorithm 1.
  $gr_{DL}(C) \leftarrow \bigcup_{r \in C} gr_{DL}(r, \mathcal{P}_{\mathcal{T}})$
  repeat
    $gr_{DL}(\mathcal{K}_n)' \leftarrow gr_{DL}(\mathcal{K}_n)$
    for all $r \in gr_{DL}(C)$ **do**
      // Delete facts from bodies of rules.
      if $r \in gr_{DL}(\mathcal{R}_s)$, $B(r) \setminus gr_{DL}(\mathcal{K}_n) = \emptyset$, and $h(r) \notin \mathcal{K}_p$ **then**
        $gr_{DL}(\mathcal{K}_n) \leftarrow gr_{DL}(\mathcal{K}_n) \cup \{h(r)\}$
        $gr_{DL}(C) \leftarrow gr_{DL}(C) \setminus \{r\}$
      end if
    end for
    // Repeat until no new facts are produced.
  until $gr_{DL}(\mathcal{K}_n) = gr_{DL}(\mathcal{K}_n)'$
  $gr_{DL}(\mathcal{R}) \leftarrow gr_{DL}(\mathcal{R}) \cup gr_{DL}(C)$
end for
// Ground assumptions and contraries as before.
$gr_{DL}(\mathcal{K}_p) \leftarrow \{b(\vec{t}) \in \mathcal{K}_p \cap \langle \mathcal{P}_{\mathcal{T}}, b \rangle\}$
$gr_{DL}(^{-}) \leftarrow \bigcup_{c \in} - gr_{DL}(c, \mathcal{P}_{\mathcal{T}})$
// Form the grounded theory.
$gr_{DL}(\mathcal{T}) \leftarrow (gr_{DL}(^{-}), gr_{DL}(\mathcal{R}_s),$
$gr_{DL}(\mathcal{R}_d), gr_{DL}(\mathcal{K}_n), gr_{DL}(\mathcal{K}_p))$
return $gr_{DL}(\mathcal{T})$

preserved, when considering the conclusions of the arguments in the extensions of both examples these coincide. In fact, the conclusions ($f(1,2)$, $b(1)$, and $a(2)$) are skeptically accepted under the complete semantics, just as in the previous example.



(a) Argumentation theory (b) Argumentation graph

Figure 3: Argumentation theory and induced AF from Example 10.

Lemma 3, and as a consequence Theorem 3, follow from the fact that the groundings obtained in the previous section and the one obtained via Algorithm 2 differ only on the simplifications on the fact base and strict rules, from whence the AFs induced by both groundings are equivalent when considering acceptable claims. The equivalence is not w.r.t. arguments since, as we have seen in Example 10, adding new facts and removing strict rules in the grounding via Algorithm 2 clearly affects the arguments that can be constructed.

Lemma 3 ($\star$)**.** *Let $\mathcal{T}$ be an argumentation theory, $grd_{DL}(\mathcal{T})$ the grounding of $\mathcal{T}$ via Algorithm 2, and $grd^2_{DL}(\mathcal{T})$ the grounding via the Datalog program $\mathcal{P}_{\mathcal{T}}$ as per Transformation 2*[7]*. Then, $\{Conc(E) \mid E \in \theta(grd^2_{DL}(\mathcal{T}))\} = \{Conc(E) \mid E \in \theta(grd_{DL}(\mathcal{T}))\}$ for $\theta \in \{com, grd, prf, stb\}$.*

Theorem 3 ($\star$)**.** *Let $\mathcal{T}$ be an argumentation theory, $grd_{DL}(\mathcal{T})$ the grounding of $\mathcal{T}$ via Algorithm 2. Then, $\{Conc(E) \mid E \in \theta(\mathcal{T})\} = \{Conc(E) \mid E \in \theta(gr(\mathcal{T}))\} = \{Conc(E) \mid E \in \theta(gr_{DL}(\mathcal{T}))\}$ for $\theta \in \{com, grd, prf, stb\}$.*

4 Evaluation

To demonstrate feasibility of our grounding approach we conducted experiments on three distinct datasets (Monterosso et al. 2025b). To evaluate the feasibility of reasoning, we used in scenario S1 F.O. ASPIC+ instances inspired by (Robaldo et al. 2024), and in scenario S2 F.O. ASPIC+ instances with increasing number of atoms, rules and variables. In scenario S3 we assess the quality of the groundings by translating well known ASP instances (Gebser, Maratea, and Ricca 2015) to ASPIC+ instances. The experiments were conducted on an Intel(R) Xeon(R) Platinum 8470, 3.5-GHz, 1.0-TB RAM. For each instance we impose a 20-minute time limit for grounding and in scenarios S1 and S2 additionally 20 minutes for reasoning. Additionally, we impose a memory limit of $32G$.

Solvers. The proposed approach (including the simplifications) has been implemented in the prototypical grounder `ANGRY` (Monterosso et al. 2025a), which is written in Rust and makes use of the Datalog engine `Nemo` (Ivliev et al. 2024). To evaluate the use of `ANGRY`as part of a ground+solve pipeline for F.O. ASPIC+ instances we combine `ANGRY` with `ASPforASPIC` (Lehtonen, Wallner, and Järvisalo 2020), an ASP-based reasoner for propositional ASPIC+. For the backend of `ASPforASPIC` we make use of `Clingo` v5.7.1. (Gebser et al. 2016). As a reference system for scenario S1 we use `Arg2P` (Calegari et al. 2022), the only argumentation tool we are aware of that handles an F.O. ASPIC+-like syntax. For scenario S3 we compare against the well-known grounder `Gringo` (Gebser et al. 2015).

Scenario S1. The primary goal is to evaluate the use of our grounding approach as part of a ground+solve strategy for reasoning over F.O. ASPIC+ instances. This involves first grounding ASPIC+ instances and in a second step solving the obtained propositional argumentation theory. More precisely, we focus on the enumeration of claim sets under the complete semantics, as this is the reasoning task the two solvers `ASPforASPIC` and `Arg2P` are capable of. The instances consist of ASPIC+ encodings denoted by Legal bench (inspired by (Robaldo et al. 2024)). We generated 20 instances with an increasing number of atoms in the knowledge base for each domain. We compare our grounder `ANGRY` together with `ASPforASPIC` against `Arg2P`. Specifically, we compare the execution time of each system to complete the enumeration of claim sets under complete semantics. Additionally, we compare the claim

[7] i.e. without the further simplifications in Algorithm 2.

sets obtained by the two system to verify the correctness of our grounding.

Scenario S2. Here we to evaluate the scalability of ANGRY. To this end we randomly generated F.O. ASPIC+ instances with increasing number of atoms, rules and variables. In each generated instance, some parameters are fixed. Constants appearing in the instances are randomly selected integers in the range $[0, 300]$. The arity of the literals, both in rules and in the atom base, is randomly chosen between 1 and 5, with a probability of 80% to fall between 1 and 3. The number of literals appearing in each rule is randomly picked from 1 to 10, with 80% probability assigned to values between 1 and 4. The number of literals involved in contrary relations is randomly selected between 1 and 3.

Based on this fixed set of parameters, we define four different configurations by varying the number of strict (str.) and defeasible (def.) rules, as well as the number of contrary relations included in the encoding. The configurations are defined as follows:

c_1: 10 rules (5 str. and 5 def.) and 7 contrary relations

c_2: 20 rules (10 str. and 10 def.) and 15 contrary relations

c_3: 40 rules (20 str. and 20 def.) and 25 contrary relations

c_4: 80 rules (40 str. and 40 def.) and 30 contrary relations

c_5: 160 rules (80 str. and 80 def.) and 90 contrary relations

For each configuration, we generate 7 sets of instances, each with a maximum number of variable symbols per rule, incrementing from 3 to 13 for each set. For every configuration and number of variable symbols, we generate 20 instances, where the number of atoms in the atom base increases from 43200 to 655584. This setup allows us to systematically analyze the scalability of ANGRY under increasing structural complexity of the instances.

Scenario S3. Here the aim is to study the extent to which our grounder meets the performance baseline of existing ASP grounders when used on ASP programs, translated to ASPIC+ theories. We compare the size of the grounding produced by Gringo in the ASP domain, with those obtained by ANGRY for the corresponding ASPIC+ instances, which are derived by translating the ASP instances used for Gringo into ASPIC+. The instances consist of three problem classes from the sixth ASP competition (Gebser, Maratea, and Ricca 2015), namely StableMarriage, VisitAll and GraphColouring. In particular, we use all of the 20 instances of each problem class. We apply the existing translations (Caminada and Schulz 2017) and (Heyninck 2019) on the ASP encodings, obtaining the argument theories given as input to ANGRY. These translations were originally designed for propositional programs, therefore we slightly adapted them to handle variables. Since the default output format for the two systems Gringo and ANGRY is different, the groundings were generated using the *text* option, which produces the groundings in a human-readable format and is very similar for both systems. Additionally, we measure and compare the grounding times of both systems. In this case, the output format used is the standard one.

Expectations.

(E1) ANGRY+ASPforASPIC will outperform Arg2P (which has struggled on larger instances in the evaluation in (Robaldo et al. 2024));

(E2) ANGRY+ASPforASPIC and Arg2P output the same claim sets for the same instance;

(E3) grounding time increases with the size of the grounding;

(E4) ANGRY can handle instances of realistic size;

(E5) ANGRY will produce slightly larger groundings compared to Gringo, due to additional rules needed for the translation from ASP to ASPIC+;

(E6) ANGRY will not outperform Gringo on ASP instances as Gringo is a very efficient system that introduces several extra optimizations for the ASP domain.

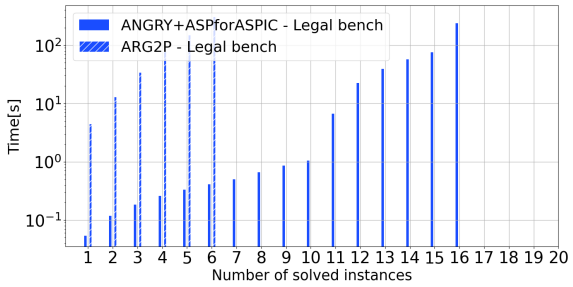


Figure 4: Results of scenario S1 comparing ANGRY+ASPforASPIC vs. Arg2P.

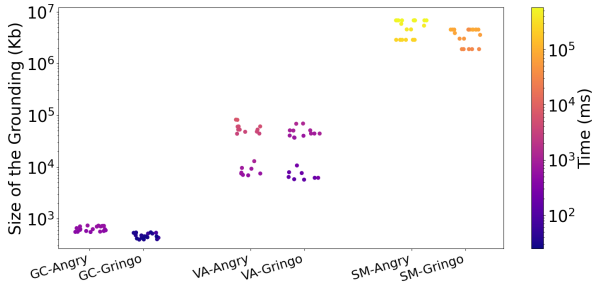


Figure 5: Results of scenario S3 comparing ANGRY vs Gringo, where GC, VA and SM stand for GraphColouring, VisitAll and StableMarriage.

Observations and Results. Figure 4 confirms (E1). We use a cactus plot, which shows the time that is at least required to solve a specific number of instances. Thus, the first bar shows the time for the fastest solving, the second bar the sum of the solving time for the two fastest instances, and so on. We report the time required by both the system ANGRY+ASPforASPIC and Arg2P to process the instances of scenario S1. From the plot, it is evident that ANGRY+ASPforASPIC outperforms Arg2P in terms of efficiency for these instances. In fact, our system solves 16 instances from LEGAL BENCH, whereas Arg2P only solves 6 instances within the time limit. For these instances we were able to confirm (E2) by assessing equivalence of the results.

Table 1 shows the results of scenario S2, where we report the number of grounded and solved instances in the given

Config		3v	4v	5v	7v	9v	11v	13v
c_1	grounded / solved	16 / 16	16 / 16	15 / 15	15 / 13	9 / 7	7 / 7	5 /4
	TO / MO	4 / 0	4 / 0	5 / 0	3 / 2	7 / 4	7 / 6	5 / 10
	mean time (s)	261.1	181.4	187.0	228.1	313.8	181.5	406.4
	median time (s)	117.5	81.8	71.3	141.9	129.7	139.6	278.4
c_2	grounded / solved	16 / 16	15 / 15	15 / 15	9 / 7	7 / 5	4 / 3	2 / 0
	TO / MO	4 / 0	5 / 0	5 / 0	9 / 2	11 / 2	9 / 7	7 / 11
	mean time (s)	284.3	259.1	336.1	418.6	488.4	580.8	731.2
	median time (s)	107.4	79.4	147.5	313.6	632.2	665.8	731.2
c_3	grounded / solved	14 / 13	13 / 13	11 / 11	5 / 4	2 / 2	2 / 2	0 / 0
	TO / MO	6 / 0	7 / 0	9 / 0	12 / 3	8 / 10	9 / 9	8 / 12
	mean time (s)	238.6	258.1	250.9	409.4	557.0	336.7	–
	median time (s)	144.7	237.9	231.7	353.1	557.0	336.7	–
c_4	grounded / solved	15 / 15	13 / 13	8 / 8	0 / 0	–	–	–
	TO / MO	5 / 0	7 / 0	9 / 3	6 / 14	–	–	–
	mean time (s)	274.6	360.2	503.4	–	–	–	–
	median time (s)	54.3	237.6	215.3	–	–	–	–
c_5	grounded / solved	10 / 7	6 / 5	3 / 3	0 / 0	–	–	–
	TO / MO	6 / 4	9 / 5	6 / 11	9 / 11	–	–	–
	mean time (s)	170.2	373.2	576.4	–	–	–	–
	median time (s)	127.3	331.7	486.9	–	–	–	–

Table 1: Results of scenario S2. Configurations $c_1, \ldots, c_5$ are as described in S2; the columns 3v, 4v, ... denote up to 3 (4, ...) variables per rule; TO stands for time out and MO for memory out, both refer to the grounding step. Mean time and median time both refer to the execution time of the grounding step, reported in seconds.

time, as well as the mean and median time for the grounding step. For those instances where the grounding step did not finish, we report if they run into a time out or ran out of memory. We can observe that in general the number of variables in the rules has the largest influence on the grounding, which is not a big surprise. Increasing the number of rules and contraries, is not a big problem, as long as the number of variables is low. For challenging instances we observe more instances failing due to memory outage than due to time out. Concluding on scenario S2 we see (E4) confirmed as `ANGRY` is able to ground instances of reasonable size and with a reasonable number of variables in rules. We assume that most real world instances would not require to use more than 4 or 5 variables within one rule.

Finally, Figure 5 confirms (E3), (E5) and (E6). There, we report the grounding size obtained by both `ANGRY` and `Gringo` for each instance from scenario S3. The grounding size obtained by `ANGRY` is, as expected, comparable and slightly larger than that by `Gringo`. This outcome confirms the quality of `ANGRY`'s grounding, as it does not contain unnecessary information. Additionally, the grounding time is reported via a color scale. As anticipated, our system is less efficient than `Gringo`. In this regard, further efforts will be made in the future to improve the system's efficiency.

5 Conclusion

We proposed a novel grounding approach for rule-based argumentation that makes use of a Datalog engine. Optimizations performed in the grounding avoid the construction of unnecessary parts and thus produces an intelligent grounding that preserves acceptable claims. Our experimental analysis demonstrates feasibility of our prototype grounder `ANGRY`, which makes use of the Datalog engine `Nemo` (Ivliev et al. 2024). In particular, the use of `ANGRY` together with the system `ASPforASPIC` (Lehtonen, Wallner, and Järvisalo 2022) as part of ground+solve approach for reasoning over ASPIC+ instances clearly outperforms the only existing argumentation-based system that we are aware of that supports a first-order ASPIC+-like syntax. On the other hand, the use of our grounder as an ASP grounder (via translations from ASP to ASPIC+ (Heyninck 2019; Caminada and Schulz 2017)) shows that our grounder meets the performance baseline of the ASP grounder `Gringo` (Gebser et al. 2015), although there is also some loss of performance due to the translations and given the additional optimizations of `Gringo` for ASP.

Future Work. We plan to extend our approach to grounding to also handle preferences, which are a further crucial aspect of frameworks for rule-based argumentation such as ABA (see e.g.(Cyras and Toni 2016; Wakaki 2017)) and ASPIC+ (Modgil and Prakken 2018). On the other hand, we want to study to what extent additional constructs that are commonly used in ASP such as conditional literals, external atoms or aggregates (Gebser and Schaub 2016) can meaningfully be incorporated. Additional efforts will also be devoted to improving the system's efficiency. This includes, evaluating the use of different Datalog and ASP (used as Datalog) grounders as the back-end of our system as well as the combination of our grounder with different reasoning approaches also for ABA (e.g. (Lehtonen, Wallner, and Järvisalo 2021; Diller, Gaggl, and Gorczyca 2022)).

Acknowledgments

This work is partially supported by BMFTR (German Federal Ministry of Research, Technology and Space) in project Semeco (Secure medical microsystems and communications) under grant 03ZU1210B. We also acknowledge support by BMFTR in DAAD project 57616814 (SECAI, School of Embedded Composite AI) as part of the program Konrad Zuse Schools of Excellence in Artificial Intelligence. Moreover, we acknowledge partial financial support from MUR project PRIN 2022 EPICA (CUP H53D23003660006) funded by the European Union - Next Generation EU and from the PNRR project FAIR - Future AI Research (PE00000013), Spoke 9 -Green-aware AI, under the NRRP MUR program funded by the NextGenerationEU. Finally, we acknowledge the computing time made available to us on the high-performance computer at the NHR Center of TU Dresden. This center is jointly supported by the BMFTR and the German state governments participating in the NHR.

References

Abiteboul, S.; Hull, R.; and Vianu, V. 1995. *Foundations of Databases*. Reading, Massachusetts: Addison-Wesley.

Besin, V.; Hecher, M.; and Woltran, S. 2023. On the structural complexity of grounding - tackling the ASP grounding bottleneck via epistemic programs and treewidth. In *ECAI*, volume 372 of *Frontiers in Artificial Intelligence and Applications*, 247–254. IOS Press.

Besnard, P.; García, A. J.; Hunter, A.; Modgil, S.; Prakken, H.; Simari, G. R.; and Toni, F. 2014. Introduction to structured argumentation. *Argument Comput.* 5(1):1–4.

Bondarenko, A.; Dung, P. M.; Kowalski, R. A.; and Toni, F. 1997. An abstract, argumentation-theoretic approach to default reasoning. *Artif. Intell.* 93:63–101.

Bondarenko, A.; Toni, F.; and Kowalski, R. A. 1993. An assumption-based framework for non-monotonic reasoning. In Pereira, L. M., and Nerode, A., eds., *Proceedings of the 2nd International Workshop on Logic Programming and Non-monotonic Reasoning (LPNMR 1993)*, 171–189. MIT Press.

Calegari, R.; Omicini, A.; Pisano, G.; and Sartor, G. 2022. Arg2P: an argumentation framework for explainable intelligent systems. *J. Log. Comput.* 32(2):369–401.

Calimeri, F.; Fuscà, D.; Perri, S.; and Zangari, J. 2017. I-DLV: the new intelligent grounder of DLV. *Intelligenza Artificiale* 11(1):5–20.

Caminada, M., and Schulz, C. 2017. On the equivalence between assumption-based argumentation and logic programming. *Journal of Artificial Intelligence Research* 60:779–825.

Ciatto, G.; Calegari, R.; and Omicini, A. 2021. 2P-Kt: A logic-based ecosystem for symbolic AI. *SoftwareX* 16:100817.

Craven, R., and Toni, F. 2016. Argument graphs and assumption-based argumentation. *Artif. Intell.* 233:1–59.

Cyras, K., and Toni, F. 2016. ABA+: assumption-based argumentation with preferences. In *KR*, 553–556.

Diller, M.; Gaggl, S. A.; and Gorczyca, P. 2021. Flexible dispute derivations with forward and backward arguments for assumption-based argumentation. In *CLAR*, volume 13040 of *LNCS*, 147–168. Springer.

Diller, M.; Gaggl, S. A.; and Gorczyca, P. 2022. Strategies in flexible dispute derivations for assumption-based argumentation. In *SAFA@COMMA*, volume 3236 of *CEUR Workshop Proceedings*, 59–72.

Dung, P. M., and Thang, P. M. 2014. Closure and consistency in logic-associated argumentation. *J. Artif. Intell. Res.* 49:79–109.

Dung, P. M. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artif. Intell.* 77(2):321–358.

East, D., and Truszczynski, M. 2000. DATALOG with constraints - an answer-set programming system. In *AAAI/IAAI*, 163–168. AAAI Press / The MIT Press.

Faber, W. 2020. An introduction to answer set programming and some of its extensions. In *RW*, volume 12258 of *Lecture Notes in Computer Science*, 149–185. Springer.

García, A. J., and Simari, G. R. 2004. Defeasible logic programming: An argumentative approach. *Theory Pract. Log. Program.* 4(1-2):95–138.

García, A. J.; Prakken, H.; and Simari, G. R. 2020. A comparative study of some central notions of ASPIC+ and DeLP. *Theory Pract. Log. Program.* 20(3):358–390.

Gebser, M., and Schaub, T. 2016. Modeling and language extensions. *AI Mag.* 37(3):33–44.

Gebser, M.; Harrison, A.; Kaminski, R.; Lifschitz, V.; and Schaub, T. 2015. Abstract gringo. *Theory and Practice of Logic Programming* 15(4-5):449–463.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Wanko, P. 2016. Theory solving made easy with clingo 5. In *Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.

Gebser, M.; Maratea, M.; and Ricca, F. 2015. The design of the sixth answer set programming competition. In Calimeri, F.; Ianni, G.; and Truszczynski, M., eds., *Logic Programming and Nonmonotonic Reasoning*, 531–544. Cham: Springer International Publishing.

Heyninck, J. 2019. Relations between assumption-based approaches in nonmonotonic logics and formal argumentation. *FLAP* 6(2):319–360.

Ivliev, A.; Gerlach, L.; Meusel, S.; Steinberg, J.; and Krötzsch, M. 2024. Nemo: Your friendly and versatile rule reasoning toolkit. In Marquis, P.; Ortiz, M.; and Pagnucco, M., eds., *Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning (KR 2024)*, 743–754. IJCAI Organization.

Jordan, H.; Scholz, B.; and Subotic, P. 2016. Soufflé: On synthesis of program analyzers. In *CAV (2)*, volume 9780 of *Lecture Notes in Computer Science*, 422–430. Springer.

Järvisalo, M.; Lehtonen, T.; and Niskanen, A. 2025. ICCMA 2023: 5th international competition on computational models of argumentation. *Artificial Intelligence* 342:104–311.

Lehtonen, T.; Rapberger, A.; Ulbricht, M.; and Wallner, J. P. 2023. Argumentation frameworks induced by assumption-based argumentation: Relating size and complexity. In *KR*, 440–450.

Lehtonen, T.; Odekerken, D.; Wallner, J. P.; and Järvisalo, M. 2024a. Complexity results and algorithms for preferential argumentative reasoning in ASPIC+. In *KR*.

Lehtonen, T.; Rapberger, A.; Toni, F.; Ulbricht, M.; and Wallner, J. P. 2024b. Instantiations and computational aspects of non-flat assumption-based argumentation. In *IJCAI*, 3457–3465. ijcai.org.

Lehtonen, T.; Wallner, J. P.; and Järvisalo, M. 2021. Declarative algorithms and complexity results for assumption-based argumentation. *J. Artif. Intell. Res.* 71:265–318.

Lehtonen, T.; Wallner, J.; and Järvisalo, M. 2022. ASPforASPIC: ASP-based algorithms for abstract rule-based argumentation (aspic+).

Lehtonen, T.; Wallner, J. P.; and Järvisalo, M. 2020. An Answer Set Programming Approach to Argumentative Reasoning in the ASPIC+ Framework. In *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning*, 636–646.

Marek, V. W., and Truszczynski, M. 1999. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm*, Artificial Intelligence. Springer. 375–398.

Modgil, S., and Prakken, H. 2018. Abstract rule-based argumentation. In Baroni, P.; Gabbay, D.; and Giacomin, M., eds., *Handbook of Formal Argumentation*. College Publications. 287–364.

Monterosso, G.; Gaggl, S. A.; Diller, M.; Hanisch, P.; and Rauschenbach, F. 2025a. ANGRY: A grounder for rule-based argumentation. https://doi.org/10.5281/zenodo.16872941.

Monterosso, G.; Gaggl, S. A.; Diller, M.; Hanisch, P.; and Rauschenbach, F. 2025b. Benchmark instances for ANGRY: A grounder for rule-based argumentation. https://doi.org/10.5281/zenodo.16875768.

Nenov, Y.; Piro, R.; Motik, B.; Horrocks, I.; Wu, Z.; and Banerjee, J. 2015. RDFox: A highly-scalable RDF store. In *ISWC (2)*, volume 9367 of *Lecture Notes in Computer Science*, 3–20. Springer.

Niemelä, I. 1999. Logic programs with stable model semantics as a constraint programming paradigm. *Ann. Math. Artif. Intell.* 25(3-4):241–273.

Odekerken, D.; Lehtonen, T.; Borg, A.; Wallner, J. P.; and Järvisalo, M. 2023. Argumentative reasoning in ASPIC+ under incomplete information. In *KR*, 531–541.

Popescu, A., and Wallner, J. P. 2023. Reasoning in assumption-based argumentation using tree-decompositions. In *JELIA*, volume 14281 of *Lecture Notes in Computer Science*, 192–208. Springer.

Prakken, H. 2010. An abstract framework for argumentation with structured arguments. *Argument Comput.* 1(2):93–124.

Robaldo, L.; Batsakis, S.; Calegari, R.; Calimeri, F.; Fujita, M.; Governatori, G.; Morelli, M. C.; Pacenza, F.; Pisano, G.; Satoh, K.; et al. 2024. Compliance checking on first-order knowledge with conflicting and compensatory norms: a comparison among currently available technologies. *Artificial Intelligence and Law* 32(2):505–555.

Urbani, J.; Jacobs, C. J. H.; and Krötzsch, M. 2016. Column-oriented datalog materialization for large knowledge graphs. In *AAAI*, 258–264. AAAI Press.

Wakaki, T. 2017. Assumption-based argumentation equipped with preferences and its application to decision making, practical reasoning, and epistemic reasoning. *Comput. Intell.* 33(4):706–736.

Weinzierl, A.; Taupe, R.; and Friedrich, G. 2020. Advancing lazy-grounding ASP solving techniques - restarts, phase saving, heuristics, and more. *Theory Pract. Log. Program.* 20(5):609–624.