Body-Decoupled Grounding via Solving: A Novel Approach on the ASP Bottleneck

Viktor Besin[1], **Markus Hecher**[1,2], **Stefan Woltran**[1]
[1]TU Wien, Vienna, Austria
[2]University of Potsdam, Potsdam, Germany
{vbesin, hecher, woltran}@dbai.tuwien.ac.at

Abstract

Answer-Set Programming (ASP) has seen tremendous progress over the last two decades and is nowadays successfully applied in many real-world domains. However, for certain problems, the well-known ASP grounding bottleneck still causes severe problems. This becomes virulent when grounding of rules, where the variables have to be replaced by constants, leads to a ground program that is too huge to be processed by the ASP solver. In this work, we tackle this problem by a novel method that decouples non-ground atoms in rules in order to delegate the evaluation of rule bodies to the solving process. Our procedure translates a non-ground normal program into a ground disjunctive program that is exponential only in the maximum predicate arity, and thus polynomial if this arity is bounded by a constant. We demonstrate the feasibility of this new method experimentally by comparing it to standard ASP technology in terms of grounding size, grounding time and total runtime.

1 Introduction

Answer set programming (ASP) [Brewka *et al.*, 2011; Gebser *et al.*, 2019; Janhunen and Niemelä, 2016] is a modeling and solving framework that can be seen as an extension of propositional satisfiability (SAT), where knowledge is expressed by means of rules comprising a *(logic) program*, whose solutions are sets of atoms, called *answer sets*, that obey every rule. Its rich first-order like language made ASP an appealing tool for modeling industrial applications (see e.g., [Falkner *et al.*, 2018]). Efficient systems are readily available ([Gebser *et al.*, 2019; Calimeri *et al.*, 2019]) and mainly build on a ground-and-solve technique, replacing variables by constants and feeding the resulting ground program into an ASP solver.

However, for certain types of problems, this ground-and-solve approach leads to the well-known ASP *grounding bottleneck* [Cuteri *et al.*, 2020; Tsamoura *et al.*, 2020], i.e., the instantiation of rules yields a program that is exponentially larger and thus too huge to be processed by the solver efficiently. We recall that the complexity for ground programs is relatively mild: consistency for disjunctive programs is located at the second level of the polynomial hierarchy, cf. [Eiter and Gottlob, 1995]; *normal programs* or *tight programs* yield NP-complete fragments [Bidoít and Froidevaux, 1991; Marek and Truszczyński, 1991]. Variables in rules increase the expressiveness and reflect the potentially huge costs of grounding: even for normal non-ground programs the complexity jumps up to NEXPTIME completeness [Dantsin *et al.*, 2001]. While the standard grounding via rule instantation leads to an exponential blow up even for programs with bounded predicate arities, there are results that indicate that the costs of grounding can be decreased when assuming such a setting. In particular, Eiter *et al.* [2007] have shown that the complexity of consistency for *non-ground normal* programs is Σ_2^P-complete. This indicates that assuming the predicate arity as fixed, there exists a polynomial translation to ground disjunctive programs, and thus an alternative grounding procedure that delegates certain efforts to the solving process.

Contributions. We provide an alternative grounding approach that utilizes the aforementioned complexity result, thereby decoupling rule bodies during grounding, which is rectified during solving. Our contributions are as follows.

1. We present a novel reduction from tight (non-ground) logic programs to disjunctive programs that encodes grounding via search, in order to identify unsatisfiable ground rules and unjustified (unfounded) atoms. In contrast to traditional grounding, our reduction allows us to *decouple* predicates occurring in the body of a rule, which might be particularly useful for larger bodies with a very dense structure.
2. We extend this approach to normal, non-ground programs, where for ensuring justifiability we additionally encode the idea of orderings (level mappings) in our reduction.
3. Finally, we present a prototype[1] that allows to translate critical parts of the program using our reduction, thereby empowering the grounding process by decoupling body predicates. Preliminary experiments indicate that this approach can lead to significant speed-ups, where the size of standard groundings would be excessively large.

Related Work. Efficient grounding is an active field of research and requires powerful rule initialization procedures trying to evade the intractable evaluation of non-ground programs. The literature distinguishes many approaches, rang-

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

ing from traditional instantiation tactics [Gebser *et al.*, 2019; Alviano *et al.*, 2019; Kaminski and Schaub, 2021] over fruitful estimations [Hippen and Lierler, 2021] and lazy grounding [Bomanson *et al.*, 2019; Weinzierl *et al.*, 2020]. Besides, there are further attempts to avoid the grounding bottleneck, like constraint-programming or ASP modulo theory extensions, e.g., [Banbara *et al.*, 2017; Janhunen *et al.*, 2017; Cabalar *et al.*, 2020], and methods based on graph invariants like treewidth [Bichler *et al.*, 2020; Calimeri *et al.*, 2019; Bliem *et al.*, 2020; Mitchell, 2019]. Similar to our work, [Eiter *et al.*, 2010] draws on the complexity of bounded arities to design space-efficient ASP evaluation methods, but their method is more in the spirit of meta-programming than on an alternative grounding procedure. Instead, we focus on decoupling body predicates and a translation to ground ASP.

2 Preliminaries

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

Ground ASP. Let ℓ, m, n be non-negative integers such that $\ell \leq m \leq n$; a_1, …, a_n be distinct propositional atoms. A *(disjunctive) program* P is a set of *(disjunctive) rules*

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

For a rule r, we let $H_r{:=}\{a_1,\ldots,a_\ell\}$, $B^+_r{:=}\{a_{\ell+1}, \ldots, a_m\}$, and $B^-_r{:=}\{a_{m+1},\ldots,a_n\}$. We denote the sets of *atoms* occurring in a rule r or in a program P by $\mathrm{at}(r){:=}H_r \cup B^+_r \cup B^-_r$ and $\mathrm{at}(\mathsf{P}){:=}\bigcup_{r\in\mathsf{P}}\mathrm{at}(r)$. A rule r is *normal* if $|H_r| \leq 1$ and a program P is *normal* if all its rules are normal. The *dependency graph* $\mathcal{D}_{\mathsf{P}}$ is the directed graph defined on the set $\bigcup_{r\in\mathsf{P}} H_r{\cup}B^+_r$ of atoms, where for every rule $r \in \mathsf{P}$ two atoms $a \in B^+_r$ and $b \in H_r$ are joined by an edge (a, b). A program P is *tight* if $\mathcal{D}_{\mathsf{P}}$ has no directed cycle [Fages, 1994].

An *interpretation* I is a set of atoms. I *satisfies* a rule r if $(H_r \cup B^-_r) \cap I \neq \emptyset$ or $B^+_r \setminus I \neq \emptyset$. I is a *model* of P if it satisfies all rules of P. The *Gelfond-Lifschitz (GL) reduct* of P under I is the program P^I obtained from P by first removing all rules r with $B^-_r \cap I \neq \emptyset$ and then removing all $\neg z$ where $z \in B^-_r$ from the remaining rules r [Gelfond and Lifschitz, 1991]. I is an *answer set* of a program P if I is a *minimal model (w.r.t.* $\subseteq$*) of* P^I. The problem of deciding whether an ASP program has an answer set is called *consistency*, which is Σ^{P}_2-complete [Eiter and Gottlob, 1995]. If the input is restricted to normal programs, the complexity drops to NP-complete [Bidoít and Froidevaux, 1991; Marek and Truszczyński, 1991]. The following characterization of answer sets is often applied for normal programs [Lin and Zhao, 2003; Janhunen, 2006]. Let I be a model of a normal program P and φ be a function (*ordering*) $\varphi : I \to \{0,\ldots,|I|-1\}$ over I. We say a rule $r \in \mathsf{P}$ is *suitable for justifying* $a \in I$ if (i) $a \in H_r$, (ii) $B^+_r \subseteq I$, (iii) $I \cap B^-_r = \emptyset$, as well as (iv) $I \cap (H_r \setminus \{a\}]) = \emptyset$. An atom $a \in I$ is *founded* if there is a rule $r \in \mathsf{P}$ *justifying* a, which is the case if r is suitable for justifying a and $\varphi(b) < \varphi(a)$ for every $b \in B^+_r$, and *unfounded* otherwise. Then, I is an *answer set* of P if (i) I is a *model of* P, and (ii) I *is founded*, i.e., every $a \in I$ is founded. For tight programs, the ordering φ is not needed.

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

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

where for every *variable vector* X_i we have $|X_i|{=}|p_i|$, and whenever $x \in \langle X_1,\ldots,X_\ell,X_{m+1},\ldots,X_n\rangle$, then $x \in \langle X_{\ell+1},\ldots,X_m\rangle$ (*safeness*). For a non-ground rule r, we let $H_r := \{p_1(X_1),\ldots,p_\ell(X_\ell)\}$, $B^+_r := \{p_{\ell+1}(X_{\ell+1}), \ldots, p_m(X_m)\}$, $B^-_r := \{p_{m+1}(X_{m+1}),\ldots,p_n(X_n)\}$, and $\mathrm{var}(r) := \{x \in X \mid p(X) \in H_r \cup B^+_r \cup B^-_r\}$. We use $\mathrm{heads}(\Pi) := \{p(X) \in H_r \mid r \in \Pi\}$, $\mathrm{hpreds}(\Pi) := \{p \mid p(X) \in \mathrm{heads}(\Pi)\}$. Without loss of generality, we assume that *variables are unique per rule*, i.e., for every two rules $r, r' \in \Pi$, we have $\mathrm{var}(r) \cap \mathrm{var}(r') = \emptyset$. Attributes *disjunctive*, *normal*, and *tight* naturally carry over to non-ground rules (programs). The rule size corresponds to $\|r\| := |B^+_r| + |B^-_r| + |H_r|$ and program size $\|\Pi\| := \sum_{r\in\Pi}\|r\|$.

In order to *ground* Π, we require a given set $\mathcal{F}$ of *facts*, i.e., atoms of the form $p(D)$ with p being a predicate of Π and D being a vector over *domain values* of size $|D| = |p|$. We say that D is part of the *domain* of Π, defined by $\mathrm{dom}(\Pi) := \{d \in D \mid p(D) \in \mathcal{F}\}$. We refer to the *domain vectors* over $\mathrm{dom}(\Pi)$ for a variable vector X of size $|X|$ by $\mathrm{dom}(X)$. Let D be a domain vector over variable vector X and vector Y contain only variables of X. We refer to the *domain vector of D restricted to Y* by D_Y. The *grounding* $\mathcal{G}(\Pi)$ consists of $\mathcal{F}$ and ground rules obtained by replacing each rule r of Form (1) for every domain vector $D \in \mathrm{dom}(\langle\mathrm{var}(r)\rangle)$ by

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

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

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

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

3 Body-Decoupled Grounding via Search

In this section, we introduce our concept of *body-decoupled* grounding, whose idea is to reduce the grounding size by decoupling dependencies between different predicates of rule bodies. We briefly motivate the potential of this idea.

Example 2. *Assume the following non-ground program* Π' *that decides in (13) for each edge (e) of a given graph, whether to pick it (p) or not ($\bar{p}$). Then, in (14) it is ensured that the choice of edges does not form triangles.*

$$p(A,B) \vee \bar{p}(A,B) \leftarrow e(A,B) \quad (13)$$
$$\leftarrow p(X,Y), p(Y,Z), p(X,Z), X \neq Y, Y \neq Z, X \neq Z. \quad (14)$$

The typical grounding effort of (14) is in $\mathcal{O}(|\mathrm{dom}(\Pi')|^3)$. *Our approach grounds body predicates of (14) individually, yielding linear bounds in the size of facts, i.e.,* $\mathcal{O}(|\mathrm{dom}(\Pi')|^2)$.

Guess Answer Set Candidates

$h(D) \vee \bar{h}(D) \leftarrow$ for every $h(X) \in \mathrm{heads}(\Pi), D \in \mathrm{dom}(X)$ (2)

Ensure Satisfiability

$\bigvee_{d \in \mathrm{dom}(x)} \mathrm{sat}_x(d) \leftarrow$ for every $r \in \Pi, x \in \mathrm{var}(r)$ (3)

$\mathrm{sat}_r \leftarrow \mathrm{sat}_{x_1}(D_{\langle x_1 \rangle}), \ldots, \mathrm{sat}_{x_\ell}(D_{\langle x_\ell \rangle}), \neg p(D)$ for every $r \in \Pi, p(X) \in B_r^+, D \in \mathrm{dom}(X), X = \langle x_1, \ldots, x_\ell \rangle$ (4)

$\mathrm{sat}_r \leftarrow \mathrm{sat}_{x_1}(D_{\langle x_1 \rangle}), \ldots, \mathrm{sat}_{x_\ell}(D_{\langle x_\ell \rangle}), p(D)$ for every $r \in \Pi, p(X) \in H_r \cup B_r^-, D \in \mathrm{dom}(X), X = \langle x_1, \ldots, x_\ell \rangle$ (5)

$\mathrm{sat} \leftarrow \mathrm{sat}_{r_1}, \ldots, \mathrm{sat}_{r_n}$ where $\Pi = \{r_1, \ldots, r_n\}$ (6)

$\mathrm{sat}_x(d) \leftarrow \mathrm{sat}$ for every $r \in \Pi, x \in \mathrm{var}(r), d \in \mathrm{dom}(x)$ (7)

$\leftarrow \neg \mathrm{sat}$ (8)

Prevent Unfoundedness

$\bigvee_{d \in \mathrm{dom}(y)} \mathrm{uf}_y(\langle D, d \rangle) \leftarrow h(D)$ for every $r \in \Pi, h(X) \in H_r, D \in \mathrm{dom}(X), y \in \mathrm{var}(r), y \notin X$ (9)

$\mathrm{uf}_r(D_X) \leftarrow \mathrm{uf}_{y_1}(D_{\langle X, y_1 \rangle}), \ldots, \mathrm{uf}_{y_\ell}(D_{\langle X, y_\ell \rangle}), \neg p(D_Y)$ for every $r \in \Pi, h(X) \in H_r, p(Y) \in B_r^+, D \in \mathrm{dom}(\langle X, Y \rangle), Y = \langle y_1, \ldots, y_\ell \rangle$ (10)

$\mathrm{uf}_r(D_X) \leftarrow \mathrm{uf}_{y_1}(D_{\langle X, y_1 \rangle}), \ldots, \mathrm{uf}_{y_\ell}(D_{\langle X, y_\ell \rangle}), p(D_Y)$ for every $r \in \Pi, h(X) \in H_r, p(Y) \in B_r^- \cup (H_r \setminus \{h(X)\}), D \in \mathrm{dom}(\langle X, Y \rangle)$, $Y = \langle y_1, \ldots, y_\ell \rangle$ (11)

$\leftarrow \mathrm{uf}_{r_1}(D), \ldots, \mathrm{uf}_{r_m}(D)$ for every $h(X) \in \mathrm{heads}(\Pi), D \in \mathrm{dom}(X), \{r_1, \ldots, r_m\} = \{r \in \Pi \mid h(X) \in H_r\}$ (12)

Figure 1: Body-decoupled grounding procedure $\mathcal{R}$ for a given tight non-ground program Π, which creates a disjunctive *ground* program.

Body-Decoupled Grounding for Tight ASP. First, we present our approach for cycle-free programs without disjunction. To this end, we assume a given non-ground, tight program Π and a set $\mathcal{F}$ of facts. For each predicate $p(X)$ in $\mathrm{heads}(\Pi)$, we use every instantiation of $p(X)$ and its negation $\bar{p}(X)$ over $\mathrm{dom}(\Pi)$, resulting in atoms $\mathrm{AtPred} := \{p(D), \bar{p}(D) \mid p(X) \in \mathrm{heads}(\Pi), D \in \mathrm{dom}(X)\}$; the atoms over $\bar{p}(X)$ are for technical reasons only, and are not needed when emplyoing e.g., choice rules [Simons *et al.*, 2002].

In addition to AtPred and in accordance with the semantics of ASP, we require to ensure (i) satisfiability and (ii) foundedness. For (i) computing models of rules, we require atoms $\mathrm{AtSat} := \{\mathrm{sat}, \mathrm{sat}_r, \mathrm{sat}_x(d) \mid r \in \Pi, x \in \mathrm{var}(r), d \in \mathrm{dom}(x)\}$, where sat ($\mathrm{sat}_r$) indicates satisfiability (of non-ground rule r), respectively. An atom of the form $\mathrm{sat}_x(d)$ indicates that for checking satisfiability, we assign variable x of non-ground rule r to domain value $d \in \mathrm{dom}(x)$. For (ii) ensuring foundedness, we use variables $\mathrm{AtUf} := \{\mathrm{uf}_r(D_X), \mathrm{uf}_y(D_{\langle X, y \rangle}) \mid r \in \Pi, D \in \mathrm{dom}(\langle \mathrm{var}(r) \rangle), h(X) \in H_r, y \in \mathrm{var}(r), y \notin X\}$. Intuitively, $\mathrm{uf}_r(D)$ indicates that r fails to justify $h(D)$ for head predicate $h(X) \in H_r$ and domain vector $D \in \mathrm{dom}(h)$, where $\mathrm{uf}_y(D_{\langle X, y \rangle})$ refers to the assigned domain value d that variable y gets in this foundedness check for $h(D)$. Overall, the number $|\mathrm{AtPred} \cup \mathrm{AtSat} \cup \mathrm{AtUf}|$ of auxiliary atoms is in $\mathcal{O}(\|\Pi\| \cdot |\mathrm{dom}(\Pi)|^{a+1})$, where a is the largest predicate arity.

Next, we are in position to explain our reduction for tight programs. The overall idea consists of three parts and is part of the reduction $\mathcal{R}$, which transforms the non-ground, tight program Π into a ground, disjunctive program $\mathcal{R}(\Pi)$ consisting of $\mathcal{F}$ and the rules given in Figure 1. Rules (2) take care of guessing answer set candidates, then Rules (3)–(8) ensure (i) satisfiability, and Rules (9)–(12) model (ii) foundedness.

Interestingly, the only disjunction that is indeed crucial and cannot be modeled via choice rules, is the disjunction part of Rules (3) for (i) satisfiability, which is responsible for guessing and saturating assignments of variables to domain values. The construction is such that whenever for a non-ground rule $r \in \Pi$, there is an assignment of variables to domain values such that the resulting ground rule is satisfied, Rules (4) or (5) yield sat_r. If such an atom sat_r can be derived for all non-ground rules of $r \in \Pi$, we follow sat by Rules (6), which is mandatory (cf. Rules (8)). Then, Rules (7) apply *saturation*, which causes the assignment of all domain values to every variable. Assuming that a grounding of a rule $r \in \Pi$ was not satisfied, then there would exist a $\subseteq$-smaller model of the reduct, invalidating the answer set candidate. Intuitively, the construction takes care that there is an answer set of $\mathcal{R}(\Pi)$, only if the grounding of Π admits an answer set.

Rules (9) are for (ii) preventing unfoundedness, ensuring that for each head atom $h(D)$ contained in a model of $\mathcal{R}(\Pi)$, variables get assigned domain values for proving foundedness. Unjustifiability of a rule $r \in \Pi$ for atom $h(D)$ is derived by Rules (10) and (11), which is prevented by Rules (12).

See Appendix for reduction $\mathcal{R}(\Pi)$ on Π from Example 1.

Theorem 1 (Correctness)**.** *Let Π be any tight, non-ground program. Then, the grounding procedure $\mathcal{R}$ on Π is correct, i.e., the answer sets of $\mathcal{R}(\Pi)$ restricted to $\mathrm{at}(\mathcal{G}(\Pi))$ match the answer sets of $\mathcal{G}(\Pi)$. Precisely, for every answer set M' of $\mathcal{R}(\Pi)$ there is an answer set $M' \cap \mathrm{at}(\mathcal{G}(\Pi))$ of $\mathcal{G}(\Pi)$.*

Proof (Sketch). $\Leftarrow$: Assume that M is an answer set of $\mathcal{G}(\Pi)$, but there is no $M' \supsetneq M$ with $M' \cap \mathrm{at}(\mathcal{G}(\Pi)) = M \cap \mathrm{at}(\mathcal{G}(\Pi))$ that is an answer set of $\mathcal{R}(\Pi)$. This results in either (i) M being not a model of $\mathcal{G}(\Pi)$, or (ii) M being unfounded. $\Rightarrow$: Let M' be an answer set of $\mathcal{R}(\Pi)$ and assume that $M := M' \cap \mathrm{at}(\mathcal{G}(\Pi))$ is not an answer set of $\mathcal{G}(\Pi)$. Then, either (i) M' is not a model of $\mathcal{R}(\Pi)$, or (ii) M' is unfounded. ☐

Notably, one can split the program and partially apply $\mathcal{R}$.

Corollary 1 (Partial Reducibility)**.** *Given a tight, non-ground program Π and a partition of Π into programs Π_1, Π_2 with $\mathrm{hpreds}(\Pi_1) \cap \mathrm{hpreds}(\Pi_2) = \emptyset$. Then, the answer sets of $\mathcal{R}(\Pi_1) \cup \mathcal{G}(\Pi_2)$ restricted to $\mathrm{at}(\mathcal{G}(\Pi))$ match those of $\mathcal{G}(\Pi)$.*

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

Improved Foundedness, replacing (10)–(12) of $\mathcal{R}$

$$\mathrm{uf}_{\mathrm{rch}(Y,X)}(D_{\langle \mathrm{rch}(Y,X)\rangle}) \leftarrow \mathrm{uf}_{y_1}(D_{\langle X,y_1\rangle}),\ldots,\mathrm{uf}_{y_\ell}(D_{\langle X,y_\ell\rangle}),\neg p(D_Y) \quad \text{for every } r \in \Pi, h(X) \in H_r, p(Y) \in B_r^+, D \in \mathrm{dom}(\langle X,Y\rangle), Y = \langle y_1,\ldots,y_\ell\rangle \tag{15}$$

$$\mathrm{uf}_{\mathrm{rch}(Y,X)}(D_{\langle \mathrm{rch}(Y,X)\rangle}) \leftarrow \mathrm{uf}_{y_1}(D_{\langle X,y_1\rangle}),\ldots,\mathrm{uf}_{y_\ell}(D_{\langle X,y_\ell\rangle}), p(D_Y) \quad \text{for every } r \in \Pi, h(X) \in H_r, p(Y) \in B_r^- \cup (H_r \setminus \{h(X)\}), D \in \mathrm{dom}(\langle X,Y\rangle), Y = \langle y_1,\ldots,y_\ell\rangle \tag{16}$$

$$\leftarrow h(D), [\bigvee_{p(Y)\in B_{r_1}} \mathrm{uf}_{\mathrm{rch}(Y,X)}(D_{\langle \mathrm{rch}(Y,X)\rangle})],\ldots,[\bigvee_{p(Y)\in B_{r_m}} \mathrm{uf}_{\mathrm{rch}(Y,X)}(D_{\langle \mathrm{rch}(Y,X)\rangle})] \quad \text{for every } h(X) \in \mathrm{heads}(\Pi), D \in \mathrm{dom}(X), \{r_1,\ldots,r_m\} = \{r \in \Pi \mid h(X) \in H_r\} \tag{17}$$

Additional Rules for Foundedness of Normal Programs

$$[p(D) \prec p'(D')] \vee [p'(D') \prec p(D)] \leftarrow \quad \text{for every } p(X), p'(X') \in \mathrm{heads}(\Pi), D \in \mathrm{dom}(X), D' \in \mathrm{dom}(X'), p(D) \neq p'(D') \tag{18}$$

$$\leftarrow [p_1(D_1) \prec p_2(D_2)], [p_2(D_2) \prec p_3(D_3)], [p_3(D_3) \prec p_1(D_1)] \quad \text{for every } p_1(X_1), p_2(X_2), p_3(X_3) \in \mathrm{heads}(\Pi), D_1 \in \mathrm{dom}(X_1), D_2 \in \mathrm{dom}(X_2), D_3 \in \mathrm{dom}(X_3), p_1(D_1) \neq p_2(D_2), p_1(D_1) \neq p_3(D_3), p_2(D_2) \neq p_3(D_3) \tag{19}$$

$$\mathrm{uf}_r(D_X) \leftarrow \mathrm{uf}_{y_1}(D_{\langle X,y_1\rangle}),\ldots, \mathrm{uf}_{y_\ell}(D_{\langle X,y_\ell\rangle}), \neg[p(D_Y) \prec h(D_X)] \quad \text{for every } r \in \Pi, h(X) \in H_r, p(Y) \in B_r^+, D \in \mathrm{dom}(\langle X,Y\rangle), Y = \langle y_1,\ldots,y_\ell\rangle, p(D_Y) \notin \mathcal{F} \tag{20}$$

Figure 2: (Top): More involved formalization of checking for unfounded atoms based on Observation 1, yielding alternative $\mathcal{R}'$ of (cf. $\mathcal{R}$ of Figure 1). (Bottom): For normal, non-ground programs, we require *additional rules* for ensuring foundedness, yielding $\mathcal{R}''$.

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

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

Proposition 1 (Disjunctive Programs Inevitable)**.** *Let Π be any tight non-ground program, where every predicate has arity at most a. Then, unless* NP $= \Sigma_2^P$*, there cannot be a polynomial grounding procedure $\mathcal{R}'$, where $\mathcal{R}'(\Pi)$ is normal.*

Proof (Idea). While consistency for normal ground programs is in NP, Σ_2^P-hardness for disjunctive (head-cycle-free) non-ground programs, cf. [Eiter *et al.*, 2007, Lem. 6], can be lifted to tight non-ground programs, by observing tightness after converting disjunctive rules into normal ones via shifting. □

Utilizing Variable Independencies. Having established the basic grounding procedure $\mathcal{R}$, we provide an improvement based on reachable paths. For a non-ground program Π, we define the *variable graph* $\mathcal{V}_\Pi$, whose vertices are the variables $\mathrm{var}(r)$ of rules $r \in \Pi$, with an edge between two $x, y \in \mathrm{var}(r)$ whenever there is a predicate $p(X)$ in r with $x, y \in X$. Let X, Y be variable vectors. We refer by $\mathrm{rch}(Y, X)$ to those vertices in X that are *reachable* by some $y \in Y$ in $\mathcal{V}_\Pi$.

Observation 1 (Variable-Justification Independency)**.** *Given a non-ground program Π, any rule $r \in \Pi$ with $h(X) \in H_r$ and $p(Y) \in B_r^+$. Further, let I be a set of atoms over $\mathcal{G}(\Pi)$. Then, if r does not justify $h(D_X) \in I$ for $D \in \mathrm{dom}(\langle X,Y\rangle)$ due to $p(D_Y) \notin I$, we have that r fails to justify* any $h(D') \in$ *I with $D' \in \mathrm{dom}(X)$ and $D'_{\langle \mathrm{rch}(Y,X)\rangle} = D_{\langle \mathrm{rch}(Y,X)\rangle}$ as well.*

Example 3. *Recall Π of Example 1. Let $I = \{b(1)\}$ and assume an arbitrary atom $a(2,1)$. The rule $a(X,Y) \leftarrow b(X), c(Y,Z)$ does not justify $a(2,1)$ due to $b(2) \notin I$. Hence, this rule fails to justify any atom $a(2,y)$ with $y \in dom(Y)$.*

Based on the observation above, we provide an alternative of reduction $\mathcal{R}$, which is preferred in case of more independent variables according to Observation 1. Instead of AtUf, we use atoms AtUfI $:= \{\mathrm{uf}_{\mathrm{rch}(Y,X)}(D_X), \mathrm{uf}_y(D_{\langle X,y\rangle}) | r \in \Pi, D \in \mathrm{dom}(\langle \mathrm{var}(r)\rangle), h(X) \in H_r, p(Y) \in B_r^+ \cup B_r^-, y \in Y, y \notin X\}$. The updated reduction $\mathcal{R}'$ is given in Figure 2 (top), where Rules (15) and (16) replace Rules (10) and (11) with the only difference that a different head predicate is used with a potentially smaller domain vector. Further, Rules (12) are replaced by Rules (17), which contain disjunctions in their bodies, as in the well-known weight rules [Gebser *et al.*, 2011]. Alternatively, one can do the cross product among the sets of disjuncts of (17).

Body-Decoupled Grounding for Normal ASP. The idea of our reduction $\mathcal{R}$ can be also lifted to normal, non-ground programs. To this end, one can rely on orderings. To simplify the presentation, we only show a variant that uses a quadratic number of auxiliary atoms for comparison, instead of encoding orderings. We therefore use for every two distinct ground atoms $p(D), p'(D)$ of Π, an additional auxiliary predicate $[p(D) \prec p'(D')]$ responsible for storing precedence in the order of derivation. Then, given $\mathcal{R}$ of Figure 1, for normal programs we just need to add those rules of Figure 2 (bottom), resulting in $\mathcal{R}''$. Intuitively, Rules (18) determine precedence among different atoms and Rules (19) take care of transitivity of "$\prec$". In addition to (15) and (16), Rules (20) add a case of unfoundedness, if precedence is not suitable for justifying.

Analogously to Theorem 1, one can show correctness of $\mathcal{R}''$ for normal, non-ground programs, including when applied to program parts as in Corollary 1. To this end, we capture predicate dependencies as follows. The *dependency graph* $\mathcal{D}_\Pi$ has as vertices the predicates $\mathrm{hpreds}(\Pi)$ with a directed edge from p to q whenever there is a rule $r \in \Pi$ with $p(X) \in B_r^+$ and $q(Y) \in H_r$. Then, a set $C \subseteq \mathrm{hpreds}(\Pi)$ of predicates is a *strongly-connected component (SCC)* if C is a $\subseteq$-largest set such that for every two distinct predicates p, q in C there is a directed path from p to q in $\mathcal{D}_\Pi$. Then, SCCs yield sufficient conditions for partially applying $\mathcal{R}''$.

Theorem 3. *Given a normal, non-ground program Π, a partition Π_1, Π_2 with $\mathrm{hpreds}(\Pi_1) \cap \mathrm{hpreds}(\Pi_2) = \emptyset$ s.t. for any SCCs C_1 of $\mathcal{D}_{\Pi_1}$, C_2 of $\mathcal{D}_{\Pi_2}$: $C_1 \cap C_2 = \emptyset$. The answer sets of $\mathcal{R}''(\Pi_1) \cup \mathcal{G}(\Pi_2)$ restricted to $\mathrm{at}(\mathcal{G}(\Pi))$ match those of $\mathcal{G}(\Pi)$.*

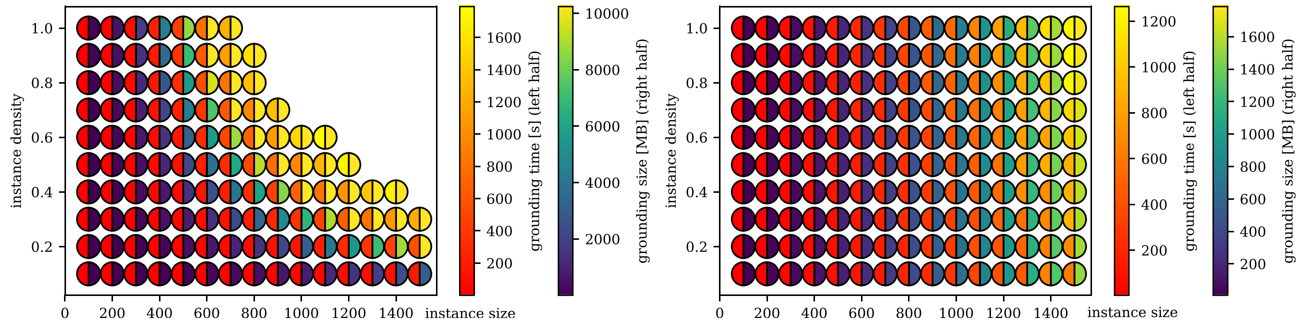

Figure 3: Grounding profile of S1 for `gringo` (left) and `newground` (right). The x-axis refers to the instance size; the y-axis indicates density. Circles mark instances grounded $< 1800s$; the left (right) half depicts grounding time (size), respectively. Mind the scales (10000 vs. 1600).

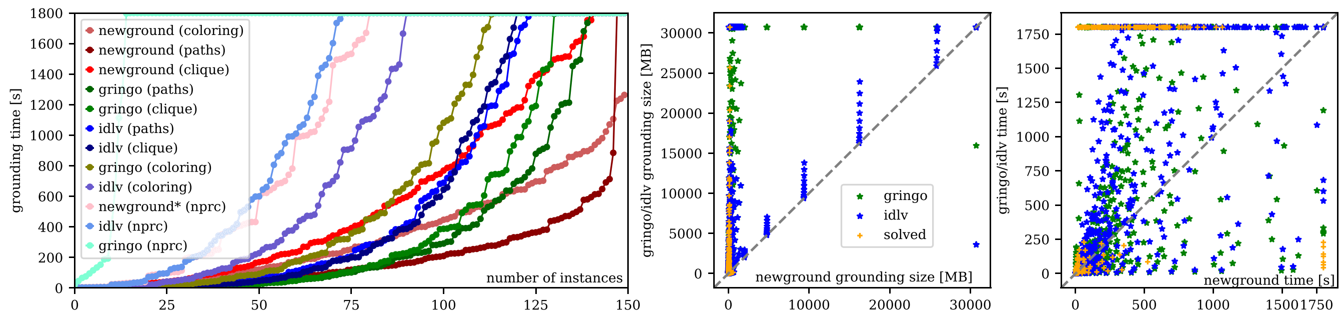

Figure 4: (Left): Cactus plot of grounding time over Scenarios S1–S4 for `newground` (red color tones), `gringo` (green tones), and `idlv` (blue tones). The x-axis shows the number of instances; the y-axis is the runtime in seconds, sorted in ascending order for each solver individually. The legend is sorted from best to worst. (Middle): Scatter plot of grounding size over Scenarios S1–S4 of `newground` (x-axis) compared to both `gringo` (blue) and `idlv` (green) on the y-axis. Those instances that could be solved are highlighted in orange. (Right): Scatter plot of grounding time over Scenarios S1–S4 of `newground` (x-axis) compared to `gringo` and `idlv` (y-axis); grounding *and solving time* in orange.

4 Implemented Prototype & Experiments

We implemented a software tool, called `newground`, realizing body-decoupled grounding via search as described above. The system `newground` is written in Python3 and uses, among others, the API and of `clingo` 5.5 and its ability to efficiently parse logic programs via syntax trees. As Corollary 1 suggests, we opted for an implementation enabling partial reducability, allowing users to select program parts that shall be reduced and others that are (traditionally) grounded. Besides Figure 1, we utilize variable independencies, as highlighted in Figure 2 (top). Internally, we optimize by pre-compiling usual compare-operators for more compact programs. Notably, `newground` is *not restricted to decision problems*, which enables counting or (quantitative) reasoning.

In order to evaluate `newground`, we design a series of benchmarks. Clearly, we cannot beat highly optimized grounders in all imaginable scenarios. Instead, we discuss potential use cases, where body-decoupled grounding is preferrable, since this approach can be incorporated into every grounder. We consider these (directed) graph *scenarios*:

S1 (coloring): Compute *edge colorings* over three colors s.t. no incoming or outgoing edges have the same color.

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

S3 (clique): Obtain directed subgraphs containing *cliques* (fully connected subgraphs of size at least three).

S4 (nprc): Compute non-partition-removal colorings, whose encoding is taken from [Weinzierl *et al.*, 2020].

S5 (stable marriage): Obtain so-called *stable marriages*; encoding taken from the ASP competition 2014.

S1 aims at providing a basic coloring problem. The decision of S2 is in P, but it can be extended; counting such paths is hard (#P-complete problem [Valiant, 1979]). Deciding S3 is polynomial-time computable, but ready to be used as part of more expressive problems. S4 (nprc) and S5 (stable marriage) are known ASP scenarios. We study *Hypotheses H1–H4*:

H1: In contrast to traditional grounding, `newground` suffers less from increased *instance density* and *instance size*.

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

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

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

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

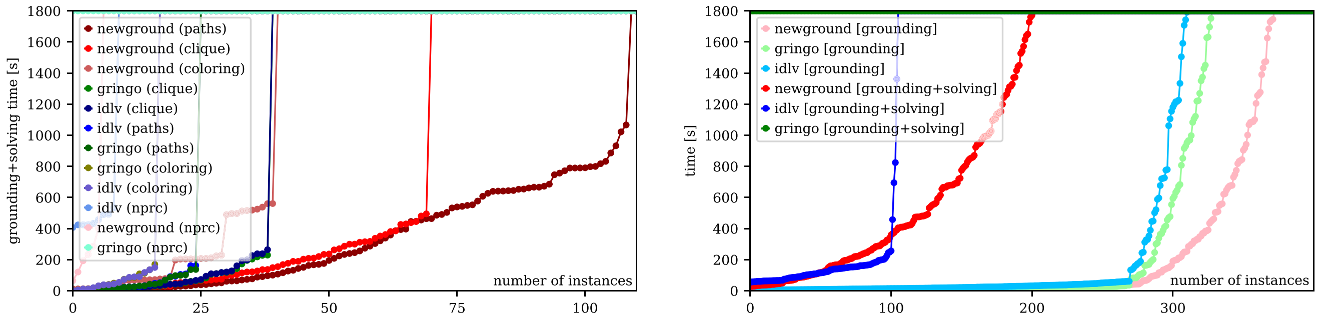

Figure 5: (Left): Corresponding cactus plot of overall (grounding *and solving*) time over Scenarios S1-S4 (cf. grounding performance of Figure 4 (Left)). (Right): Cactus plot of grounding time as well as grounding and solving performance for Scenario S5 (stable marriage).

Benchmark Setting. We only study the performance of the following *exact (full) grounders* (e.g., no lazy grounding).

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

We *measure grounding sizes*, *grounding times*, and solving capabilities of groundings. For full intercomparability, grounding size measures the size of the corresponding text output without I/O operations. The reason is that `newground` can be interleaved with other program parts, which fails to work with smodels or aspif formats out-of-the-box. Anyway, relative orders of magnitude of measured grounding sizes unambiguously stand. For comparing *overall performance*, we use solver `clingo` 5.5.1 with "-q --stats=2", i.e., we compute one answer set; for `newground` we add "--project" to ensure answer sets are over the same atoms. We *limit* main memory (RAM) to 16GB and overall runtimes (grounding *& solving*) to 1800s. Plots use *cut-off* grounding sizes of 10GB or 30GB.

Results: Grounding Scalability Study. For systematically studying the grounding of S1–S5, we use crafted instances and create for each solver a grounding profile. Figure 3 depicts the grounding profiles of S1 for `gringo` (left) and `newground` (right), showing grounding times and sizes, depending on the instance size (x-axis) and density (y-axis). Interestingly, for `newground` grounding times and sizes for a fixed instance size (column) of Figure 3 (right) are quite similar. This is in contrast to Figure 3 (left), suggesting that compared to `gringo`, `newground` is not that sensitive to instance density. Further, the rows of Figure 3 (right) are also similar. Overall, we confirm H1 (see Appendix for more data).

Results: Grounding Performance. In Figure 4 (left), we show a cactus plot of grounding times over S1–S4 for all grounders. Interestingly, `newground` clearly outperforms here; `gringo` performs second-best, except for S4 (nprc), where treewidth-based grounding seems promising. The grounding sizes over all instances for S1–S4 are given in the scatter plot Figure 4 (middle), which shows that `newground` massively reduces sizes (almost all dots above diagonal). Interestingly, `newground` *solves* instances, where `gringo` and `idlv` output groundings beyond 30GB. Figure 4 (right) shows a scatter plot revealing that `newground` improves grounding times (blue/green dots above diagonal). Among those below the diagonal, most dots are below 250s (y-axis), suggesting a portfolio that uses `gringo` or `idlv` for 250s and switches to `newground` if unsuccessful. The orange dots represent overall runtimes among solved instances, showing that no instance below the diagonal (where `newground` falls behind) can be solved in more than 250s. Overall, we confirm H2.

Results: Overall Performance. For a deeper study of overall (solving) performance, we refer to the cactus plot of Figure 5 (left), showing runtimes for S1–S4. While `newground` performs best, we still see a clear difference between solving and grounding performance (cf. Figure 4). One could believe that analyzing combined grounding and solving is unfair for `gringo` and `idlv`, since `newground` grounds faster. However, Figure 4 (left) shows that for, e.g., S2 and S3, there are many instances grounded by `gringo` within 200s. Interestingly, Figure 5 (left) reveals that only a small amount of those can then actually be solved by `clingo` within the remaining 1600s. This is also visible in Figure 5 (right), which depicts a cactus plot for S5 over grounding as well as overall runtime. We expect potential to optimize `clingo`'s internal handling of large programs (big data) for better utilizing `newground`'s grounding performance. However, we confirm H3; and H4 due to the integration of `newground` with `gringo` and `idlv`.

5 Conclusion, Discussion & Future Work

This work introduces a technique on body-decoupled grounding, where grounding-intense ASP rules are treated by means of a novel reduction. The reduction translates tight (normal) non-ground rules into disjunctive ground rules, thereby being exponential only in the maximum predicate arity. Our empirical evaluation shows an advantage: Compared to state-of-the-art exact grounders, *body-decoupled* grounding applied on crucial program parts massively reduces grounding size.

We plan on integrating our approach into (intelligent) grounders, aiming for heuristics that automatically estimate program parts where `newground` likely pays off. Further, our approach could be extended to disjunctive non-ground programs, which unfortunately is Σ_3^P-complete for bounded arity [Eiter *et al.*, 2007]. However, we expect "almost tightness" notions, e.g., [Fandinno and Hecher, 2021; Hecher, 2022], to aid in reducing heavy rules for disjunctive atoms.

Acknowledgments

This research has been supported by the Vienna Science and Technology Fund (WWTF) grant ICT19-065, and by the Austrian Science Fund (FWF) grants P32830 and Y698.

References

[Alviano *et al.*, 2019] Mario Alviano, Giovanni Amendola, Carmine Dodaro, Nicola Leone, Marco Maratea, and Francesco Ricca. Evaluation of Disjunctive Programs in WASP. In *LPNMR*, volume 11481 of *LNCS*, pages 241–255. Springer, 2019.

[Banbara *et al.*, 2017] Mutsunori Banbara, Benjamin Kaufmann, Max Ostrowski, and Torsten Schaub. Clingcon: The next generation. *Theory Pract. Log. Program.*, 17(4):408–461, 2017.

[Bichler *et al.*, 2020] Manuel Bichler, Michael Morak, and Stefan Woltran. lpopt: A Rule Optimization Tool for Answer Set Programming. *Fundamenta Informaticae*, 177(3-4):275–296, 2020.

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

[Bliem *et al.*, 2020] Bernhard Bliem, Michael Morak, Marius Moldovan, and Stefan Woltran. The Impact of Treewidth on Grounding and Solving of Answer Set Programs. *Journal of Artificial Intelligence Research*, 67:35–80, 2020.

[Bomanson *et al.*, 2019] Jori Bomanson, Tomi Janhunen, and Antonius Weinzierl. Enhancing lazy grounding with lazy normalization in answer-set programming. In *AAAI'19*, pages 2694–2702. AAAI Press, 2019.

[Brewka *et al.*, 2011] G. Brewka, T. Eiter, and M. Truszczyński. Answer set programming at a glance. *Comm. of the ACM*, 54(12):92–103, 2011.

[Cabalar *et al.*, 2020] Pedro Cabalar, Jorge Fandinno, Torsten Schaub, and Philipp Wanko. A uniform treatment of aggregates and constraints in hybrid ASP. In *KR*, pages 193–202, 2020.

[Calimeri *et al.*, 2019] Francesco Calimeri, Simona Perri, and Jessica Zangari. Optimizing answer set computation via heuristic-based decomposition. *Theory Pract. Log. Program.*, 19(4):603–628, 2019.

[Cuteri *et al.*, 2020] Bernardo Cuteri, Carmine Dodaro, Francesco Ricca, and Peter Schüller. Overcoming the grounding bottleneck due to constraints in ASP solving: Constraints become propagators. In *IJCAI*, pages 1688–1694. ijcai.org, 2020.

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

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

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

[Eiter *et al.*, 2010] Thomas Eiter, Wolfgang Faber, and Mushthofa Mushthofa. Space efficient evaluation of ASP programs with bounded predicate arities. In *AAAI'10*. AAAI Press, 2010.

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

[Falkner *et al.*, 2018] Andreas A. Falkner, Gerhard Friedrich, Konstantin Schekotihin, Richard Taupe, and Erich Christian Teppan. Industrial applications of answer set programming. *Künstliche Intell.*, 32(2-3):165–176, 2018.

[Fandinno and Hecher, 2021] Jorge Fandinno and Markus Hecher. Treewidth-Aware Complexity in ASP: Not all Positive Cycles are Equally Hard. In *AAAI*, pages 6312–6320. AAAI Press, 2021.

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

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

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical Negation in Logic Programs and Disjunctive Databases. *New Generation Computing*, 9(3/4):365–386, 1991.

[Hecher, 2022] Markus Hecher. Treewidth-aware reductions of normal ASP to SAT - Is normal ASP harder than SAT after all? *Artif. Intell.*, 304:103651, 2022.

[Hippen and Lierler, 2021] Nicholas Hippen and Yuliya Lierler. Estimating grounding sizes of logic programs under answer set semantics. In *JELIA*, volume 12678 of *LNCS*, pages 346–361. Springer, 2021.

[Janhunen and Niemelä, 2016] Tomi Janhunen and Ikka Niemelä. The Answer Set Programming Paradigm. *AI Magazine*, 37(3):13–24, 2016.

[Janhunen *et al.*, 2017] Tomi Janhunen, Roland Kaminski, Max Ostrowski, Sebastian Schellhorn, Philipp Wanko, and Torsten Schaub. Clingo goes linear constraints over reals and integers. *Theory Pract. Log. Program.*, 17(5-6):872–888, 2017.

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

[Kaminski and Schaub, 2021] Roland Kaminski and Torsten Schaub. On the foundations of grounding in answer set programming. *CoRR*, abs/2108.04769, 2021.

[Lin and Zhao, 2003] Fangzhen Lin and Jicheng Zhao. On tight logic programs and yet another translation from normal logic programs to propositional logic. In *IJCAI'03*, pages 853–858. Morgan Kaufmann, 2003.

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

[Mitchell, 2019] David Mitchell. Guarded constraint models define treewidth preserving reductions. In *CP*, volume 11802 of *LNCS*, pages 350–365. Springer, 2019.

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

[Tsamoura *et al.*, 2020] Efthymia Tsamoura, Víctor Gutiérrez-Basulto, and Angelika Kimmig. Beyond the grounding bottleneck: Datalog techniques for inference in probabilistic logic programs. In *AAAI'20*, pages 10284–10291. AAAI Press, 2020.

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

[Weinzierl *et al.*, 2020] Antonius Weinzierl, Richard Taupe, and Gerhard Friedrich. Advancing lazy-grounding ASP solving techniques - restarts, phase saving, heuristics, and more. *Theory Pract. Log. Program.*, 20(5):609–624, 2020.