FastFound: Easing the ASP Bottleneck via Predicate-Decoupled Grounding

Alexander Beiser[1] , **Martin Gebser**[2] , **Markus Hecher**[3] , **Stefan Woltran**[1]
[1]TU Wien, Vienna, Austria
[2]University of Klagenfurt, Klagenfurt, Austria
[3]CNRS, Computer Science Research Center of Lens (CRIL), Univ. Artois, Lens, France
{alexander.beiser,stefan.woltran}@tuwien.ac.at, martin.gebser@aau.at, hecher@cril.fr

Abstract

The grounding bottleneck in Answer Set Programming prohibits large instances from being solved. This is caused by a combinatorial explosion in the grounding phase of standard *ground&solve* systems. A promising alternative is Body-Decoupled Grounding (BDG), which grounds each body predicate on its own. However, BDG faces challenges in terms of worst-case grounding size and limited interoperability with other systems.

This paper addresses shortcomings of BDG by introducing FastFound: an alternative foundedness check that significantly reduces grounding sizes, by grounding *each predicate* on its own. FastFound's foundedness check is done implicitly, which leads to a quadratic reduction in grounding size. We start by introducing FastFound for tight normal rules, where we observe that this cannot be substantially improved. Then we extend FastFound to head-cycle-free programs and give novel interoperability results for full disjunctive programs. An experimental evaluation on our prototype shows promising results, as we solve more grounding-heavy tasks than both standard ground&solve systems and BDG.

1 Introduction

Answer Set Programming (ASP) (Gelfond and Leone 2002) is a logic programming paradigm with various applications, both in industry (Falkner et al. 2018) and science (Erdem, Gelfond, and Leone 2016). Solutions to ASP programs are called answer sets. Automatic computation of answer sets is enabled by modern efficient systems like `clingo` (Gebser et al. 2016) and `dlv` (Leone et al. 2006). These approaches are based on the so-called *ground&solve* paradigm, which proceeds by first grounding, instantiating variables, and then solving, computing answers by SAT-like solvers.

Although modern grounders, like `gringo` (Gebser et al. 2015) and `idlv` (Calimeri et al. 2017) are highly optimized systems, they still suffer from the so-called *grounding bottleneck* (Gebser et al. 2018). Resulting from the combinatorial explosion in the variable instantiation phase, the worst-case grounding size is exponential in the number of variables of a program. Alternative grounding procedures, such as lazy grounding (Weinzierl 2017), compilation-based techniques (Dodaro, Mazzotta, and Ricca 2024), or Body-Decoupled Grounding (BDG) (Besin, Hecher, and Woltran 2022; Beiser et al. 2024), partially alleviate the problem. BDG is particularly promising on grounding-heavy problems. It shifts effort from the grounder to the solver, by decomposing rules into their predicates and grounding the body predicates one by one. The resulting grounding size is exponential in two times the maximum predicate arity, whereas groundings of standard grounders are exponential in the number of rule variables. Compared to standard grounders, the effort in terms of solving increases by one level of the polynomial hierarchy:

Method	Grounding Effort	Solving Effort
Standard	$\approx \lvert\mathrm{dom}(\Pi)\rvert^{\lvert\mathrm{var}(\Pi)\rvert}$	normal ASP
BDG	$\approx \lvert\mathrm{dom}(\Pi)\rvert^{2\cdot a}$	disjunctive ASP
FastFound	$\approx \lvert\mathrm{dom}(\Pi)\rvert^{a+1}$	disjunctive ASP

Comparison of grounding methods. Standard grounding is exponential in the program size, whereas the size of BDG depends on the largest predicate arity a. Our approach avoids a quadratic blow-up, which is essentially optimal.

The quadratic blow-up of BDG is particularly problematic in practical solving. It originates from foundedness, where one must keep track of every instantiation of head predicate (variables) and link this to a corresponding instantiation of a body predicate, which might double the involved arities. Further, interoperability with other approaches has been limited to head-cycle-free programs, which might be problematic for some practical problems.

Contributions. We address BDG's shortcomings as follows:

1. We first introduce FastFound for tight normal programs, providing an alternative foundedness check for BDG that has a quadratic improvement in grounding size. It turns out that we cannot substantially improve this bound. We then proceed to extend FastFound to head-cycle-free programs and the non-tight case.
2. Further, we obtain interoperability results on full disjunctive programs, which allows us to interleave standard grounding with FastFound.
3. We prototypically integrate FastFound into `newground` (Besin, Hecher, and Woltran 2022). There, we show promising results of FastFound on grounding-heavy problems in terms of both grounding and solving performance.

Related Work. Standard grounding (Kaminski and Schaub

2023) implements bottom-up/semi-naive grounding, which grounds a program along its dependency graph (Gebser, Kaminski, and Schaub 2015). The modern grounders `gringo` (Gebser et al. 2015) and `idlv` (Calimeri et al. 2017) implement semi-naive grounding. Lazy grounding (Weinzierl 2017; Leutgeb and Weinzierl 2018; Bomanson, Janhunen, and Weinzierl 2019) skips the separate grounding phase and interleaves grounding with solving. Compilation-based approaches skip grounding, by compiling parts of a program s.t. this compiled program injects constraints (Cuteri et al. 2019; Lierler and Robbins 2021) or additional nogoods (Mazzotta, Ricca, and Dodaro 2022; Dodaro, Mazzotta, and Ricca 2024) during solving. ASP modulo theories integrates approaches like constraint programming (Banbara et al. 2017) or mixed-integer programming (Liu, Janhunen, and Niemela 2012). This has also been tackled by skipping grounding altogether—using s(CASP), which has no preliminary grounding phase (Arias et al. 2018). In this paper we introduce FastFound, which is orthogonal to the previously mentioned approaches, by rewriting and decoupling non-ground rules.

Structure-aware techniques ground bag by bag on a minimal tree decomposition of a rule's variable graph (Bichler, Morak, and Woltran 2016), while FastFound grounds predicate by predicate. BDG (Besin, Hecher, and Woltran 2022) relies on a basic technique for handling foundedness of atoms, which we improve with FastFound. Hybrid Grounding (Beiser et al. 2024) enables interoperability of BDG with head-cycle-free rules, which we extend to full disjunctive programs. Recent work (Besin, Hecher, and Woltran 2023) shows a reduction from full disjunctive ASP to epistemic logic programming, while FastFound stays in ASP.

Structure. We define preliminaries below. Section 3 introduces the FastFound reduction, and Section 4 shows how FastFound is interoperable with full disjunctive programs. After experimentally demonstrating FastFound in Section 5, we conclude the paper in Section 6.

2 Preliminaries

Ground ASP. An ASP program Π is a set of rules $r \in \Pi$: $a_1 \vee \ldots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, \neg a_{m+1}, \ldots, \neg a_n$, where $l, m, n \in \mathbb{N}$ s.t. $l \leq m \leq n$ and $a_1, \ldots, a_n$ are propositional atoms. Let $H_r = \{a_1, \ldots, a_l\}$, $B_r^+ = \{a_{l+1}, \ldots, a_m\}$, $B_r^- = \{a_{m+1}, \ldots, a_n\}$, and $B_r = B_r^+ \cup B_r^-$. We say that $r \in \Pi$ is *normal* iff $|H_r| \leq 1$, a *constraint* iff $|H_r| = 0$, and *disjunctive* iff $|H_r| > 1$. Choice rules are defined as usual (Calimeri et al. 2020). Π is *disjunctive* iff it contains at least one disjunctive rule, otherwise Π is *normal*. The (positive) *dependency graph* $\mathcal{D} = (V, E)$ is a directed graph, where $V = \bigcup_{r \in \Pi}(H_r \cup B_r^+)$ and $E = \{(b, h) \mid r \in \Pi, b \in B_r^+, h \in H_r\}$. Program Π is *tight* iff there is no cycle in $\mathcal{D}$, and Π is *head-cycle-free (HCF)* iff there is no cycle in $\mathcal{D}$ among two atoms $\{a, b\} \subseteq H_r$ for any $r \in \Pi$. Let $\mathrm{SCC}(\Pi)$ be the *strongly connected components (SCCs)* of $\mathcal{D}$, let ϵ be a fresh constant, and $\mathrm{SCC}(\Pi, p) = \{v \in S \mid S \in \mathrm{SCC}(\Pi), p \in S\} \cup \{\epsilon \mid S \in \mathrm{SCC}(\Pi), p \in S, v \in S, (v, v) \in E\}$.

Let $\mathrm{HB}(\Pi)$ be the set of all atoms of Π. An *interpretation* $I \subseteq \mathrm{HB}(\Pi)$ is a set of (true) atoms. I *satisfies* a rule r iff $(H_r \cup B_r^-) \cap I \neq \emptyset$ or $B_r^+ \setminus I \neq \emptyset$. I is a *model* of Π iff it satisfies all rules of Π. The *Gelfond-Lifschitz (GL) reduct* of Π under I is the program Π^I obtained from Π by first removing all rules r with $B_r^- \cap I \neq \emptyset$ and then removing all $\neg a$ where $a \in B_r^-$ from the remaining rules r (Gelfond and Lifschitz 1991). I is an *answer set* of a program Π iff I is a *minimal model* (w.r.t. $\subseteq$) of P^I. For an interpretation I, a *level mapping* $\varphi : I \to \{0, \ldots, |I| - 1\}$ assigns every atom in I a unique value (Lin and Zhao 2003; Janhunen 2006). Let I be an interpretation of a normal (HCF) program Π. An atom $a \in I$ is *founded* iff there is a rule $r \in \Pi$ s.t. (i) r is suitable for justifying a, i.e., $H_r \cap I = \{a\}$, $B_r^+ \subseteq I$, and $B_r^- \cap I = \emptyset$, and (ii) there are no cyclic-derivations, i.e., $\forall b \in B_r^+ : \varphi(b) < \varphi(a)$. I is an *answer set* of a normal (HCF) program Π iff I is a model of Π s.t. all atoms in I are founded for some level mapping.

Non-ground ASP. A non-ground ASP program Π is a set of rules $r \in \Pi$: $p_1(\mathbf{X}_1) \vee \ldots \vee p_\ell(\mathbf{X}_\ell) \leftarrow p_{\ell+1}(\mathbf{X}_{\ell+1}), \ldots, p_m(\mathbf{X}_m), \neg p_{m+1}(\mathbf{X}_{m+1}), \ldots, \neg p_n(\mathbf{X}_n)$, where $l, m, n \in \mathbb{N}$ s.t. $l \leq m \leq n$ and $p_1(\mathbf{X}_1), \ldots, p_n(\mathbf{X}_n)$ are predicates. For a predicate $p_i(\mathbf{X}_i)$, p_i is its predicate name, and $\mathbf{X}_i = \langle x_1, \ldots, x_a \rangle$ is its term vector. A *term* is a constant or a variable, and $|p_i| = |\mathbf{X}_i| = a$ is the arity of $p_i(\mathbf{X}_i)$. By $x \in \mathbf{X}$, we denote that x *is in* $\mathbf{X}$. We require variable safeness, define H_r, B_r, B_r^+, and B_r^- analogously to the ground case, and let $\mathrm{pred}(r) = \{p \mid p(\mathbf{X}) \in r\}$, $\mathrm{pred}(\Pi) = \bigcup_{r \in \Pi} \mathrm{pred}(r)$, $\mathrm{var}(r) = \{x \in \mathbf{X} \mid p(\mathbf{X}) \in B_r^+\}$, $\mathrm{var}(\Pi) = \bigcup_{r \in \Pi} \mathrm{var}(r)$. Wlog., we assume that variables are unique per rule, i.e., $\mathrm{var}(r_1) \cap \mathrm{var}(r_2) = \emptyset$ for any two $r_1, r_2 \in \Pi$. The attributes *disjunctive*, *normal*, *constraint*, and *tight (HCF)* carry over to non-ground rules (programs), where we define the dependency graph $\mathcal{D}$ over $\mathrm{pred}(\Pi)$. For a predicate $p \in \mathrm{pred}(\Pi)$, let $\mathrm{RULES}(p) = \{r \in \Pi \mid p \in \mathrm{pred}(H_r)\}$. Grounding refers to the instantiation of variables by their domain. Let $\mathcal{F}_p = \{p(\mathbf{D}) \mid r \in \Pi, \mathrm{var}(r) = \emptyset, B_r = \emptyset, p(\mathbf{D}) \in H_r\}$ be the fact predicates, $\mathrm{dom}(\Pi) = \{d \in \mathbf{D} \mid p(\mathbf{D}) \in \mathcal{F}_p\}$ be the domain, and let wlog. $\mathrm{dom}(x) = \mathrm{dom}(\Pi)$ be for any variable x. The instantiations $\mathrm{dom}(\mathbf{X})$ of a variable vector $\mathbf{X} = \langle x_1, \ldots, x_k \rangle$ contain all term vectors $\mathbf{D} = \langle d_1, \ldots, d_k \rangle$ with $d_1 \in \mathrm{dom}(x_1), \ldots, d_k \in \mathrm{dom}(x_k)$. By $\mathbf{D}_{\langle \mathbf{Y} \rangle} = \langle d_i, \ldots, d_j \rangle$, we denote the restriction of $\mathbf{D} \in \mathrm{dom}(\mathbf{X})$ to a variable vector $\mathbf{Y} = \langle x_i, \ldots, x_j \rangle$ s.t. $\{x_i, \ldots, x_j\} \subseteq \{x_1, \ldots, x_k\}$. A rule $r \in \Pi$ over the variables $\mathbf{X}$ is naively grounded by $\mathcal{G}(r) = \{p_1(\mathbf{D}_{\langle \mathbf{X}_1 \rangle}) \vee \ldots \vee p_l(\mathbf{D}_{\langle \mathbf{X}_l \rangle}) \leftarrow p_{l+1}(\mathbf{D}_{\langle \mathbf{X}_{l+1} \rangle}), \ldots, p_m(\mathbf{D}_{\langle \mathbf{X}_m \rangle}), \neg p_{m+1}(\mathbf{D}_{\langle \mathbf{X}_{m+1} \rangle}), \ldots, \neg p_n(\mathbf{D}_{\langle \mathbf{X}_n \rangle}) \mid \mathbf{D} \in \mathrm{dom}(\mathbf{X})\}$. A program is grounded (naively) by $\mathcal{G}(\Pi) = \bigcup_{r \in \Pi} \mathcal{G}(r)$.

The size of a rule r is $|r| = |H_r \cup B_r|$, and $|\Pi| = \sum_{r \in \Pi} |r|$ for a program Π. A naively grounded rule r has a *grounding size* in $\mathcal{O}\left(|r| \cdot |\mathrm{dom}(\Pi)|^{|\mathrm{var}(r)|}\right)$. For brevity, we write $\approx |\mathrm{dom}(\Pi)|^{|\mathrm{var}(r)|}$. The grounding size of a program Π is $\approx |\mathrm{dom}(\Pi)|^{\max_{r \in \Pi} |\mathrm{var}(r)|}$, or $\approx |\mathrm{dom}(\Pi)|^{|\mathrm{var}(\Pi)|}$ in the worst-case. The semantics of a non-ground program Π is defined over $\mathcal{G}(\Pi)$, where the Herbrand base is $\mathrm{HB}(\Pi) = \{p(\mathbf{D}) \in H_r \cup B_r \mid r \in \mathcal{G}(\Pi)\}$. Given that the worst-case grounding size of state-of-the-art (SOTA-)grounding

matches naive grounding $\mathcal{G}(\Pi)$ (Kaminski and Schaub 2023), we use the terms bottom-up, semi-naive, SOTA-, and naive grounding interchangeably.

Complexity Theory. We assume familiarity with notions of complexity theory. Given a program Π, the answer set existence problem asks whether Π has an answer set[1]. For ground ASP, it is Σ_2^{P}-complete (Eiter and Gottlob 1995), which drops to NP-completeness for HCF programs (Dechter 1994). The complexity increases to $\mathtt{NEXPTIME}^{\mathtt{NP}}$-completeness for non-ground programs and to NEXPTIME-completeness for non-ground HCF ASP (Dantsin, Eiter, and Gottlob 2001). Assuming bounded predicate arities, it drops to Σ_2^{P}-completeness for HCF/normal programs, and Σ_3^{P}-completeness for full disjunctive programs (Eiter et al. 2007).

3 Quadratic Boost via FastFound

First, we recall the challenges that BDG faces.

3.1 Shortcomings of BDG

BDG grounds non-ground normal programs with a translation to ground disjunctive programs. This translation (1) ensures satisfiability of all rules and (2) prevents unfoundedness of atoms. Let a be the maximum predicate arity. BDG ignores unfoundedness for constraints, as there is no head atom. We are left with the satisfiability check, which is performed in $\approx |\operatorname{dom}(\Pi)|^a$. However, normal rules require prevention of unfoundedness, which increases grounding size to $\approx |\operatorname{dom}(\Pi)|^{2\cdot a}$ (Besin, Hecher, and Woltran 2022).

Example 1. *Consider the listing below. We are given a graph instance defined by predicate* e*, where we assume a simple graph. First we guess two subgraphs defined by predicates* f *and* d*. We denote the two rules in Line 1 as* $\mathbf{r}_1$*. For* f*, we require that no clique of size ≥3 exists and for* d*, we want to derive all vertices that are part of a clique of size ≥3. These shall be gathered in predicate* c *for later processing. Let the rules in Lines 2–3 be* r_2 *and* r_3*, respectively.*

```
1 {f(X,Y)} ← e(X,Y). {d(X,Y)} ← e(X,Y).
2 ← f(X1,X2), f(X1,X3), f(X2,X3).
3 c(X1) ← d(X1,X2), d(X1,X3), d(X2,X3).
```

Standard grounding has a grounding size $\approx |\operatorname{dom}(\Pi)|^3$ *for both* r_2 *and* r_3*. BDG's grounding size is* $\approx |\operatorname{dom}(\Pi)|^2$ *for* r_2*, and* $\approx |\operatorname{dom}(\Pi)|^3$ *for* r_3*.*

Observe the cubic grounding sizes of BDG and SOTA-grounding for r_3*. As BDG pushes effort from grounding to solving, its overall performance is worse than SOTA-grounding, which renders BDG useless for* r_3*.*

3.2 FastFound: Efficient Foundedness

In this section, we first introduce FastFound for tight normal programs Π. Later we will extend this to ensure interoperability with standard grounding procedures. Compared to the *quadratic increase* in grounding size of BDG, we define a reduction avoiding it. We obtain linear optimal groundings, which is a *worst-case limit* (assuming answer sets contain all ground atoms of a program).

[1]The answer set existence problem considers both facts (data) and the encoding (combined complexity).

Observation 1 (LB)**.** *For any predicate arity* $a \geq 0$*, there is a non-ground program* Π *such that its answer set contains all atoms in HB*(Π) *and is of size* $\Omega(|\Pi| \cdot |\operatorname{dom}(\Pi)|^a)$*.*

Proof. Construct such a *worst-case* non-ground program Π of arity a, where each predicate occurs constantly often and the answer set of Π is of size $\Omega(|\Pi| \cdot |\operatorname{dom}(\Pi)|^a)$, containing all ground atoms. □

Fastfound approaches this bound up to $O(|\Pi| \cdot |\operatorname{dom}(\Pi)|^{a+1})$ for normal programs, by splitting grounding into three parts:

1. Guessing ground atoms of the answer set candidate,
2. Checking that every (ground) rule is indeed satisfied, and
3. Ensuring that every guessed atom is founded: This part requires care, as naive encodings cause a quadratic increase, which we avoid via saturation[2].

Figure 1 presents the reduction from a tight normal program to a disjunctive program, where parts of the grounding effort are shifted from the grounder to the solver. The three parts are represented in the figure as follows. True atoms are guessed by Equations (2)–(3), satisfiability is ensured by Equations (4)–(9), and foundedness is given by Equations (10)–(19).

1. Guessing Atoms. By decoupling the rule structure, we ground the head independent of its body. We introduce auxiliary predicates that enable partitioning a program Π into a part $\Pi_{\mathcal{G}}$ that is grounded by any grounding approach in Equation (1), and a part $\Pi_{\mathcal{H}}$ grounded by FastFound in Equations (2)–(19).

Example 2. *We proceed by showing FastFound's grounding of* r_3 *from Example 1. Let us assume that* $\operatorname{dom}(\Pi) = \operatorname{dom}(X1) = \operatorname{dom}(X2) = \operatorname{dom}(X3) = \{1, 2\}$*. First, we guess whether the auxiliary head holds. The following listing provides the result of applying Equations (2)–(3).*

```
1 {c'(1)}. {c'(2)}. c(1) ← c'(1). c(2) ← c'(2).
```

2. Satisfiability. For an answer set I of Π and for any non-ground rule $r \in \Pi$, all of r's ground instantiations must be satisfied. This is encoded with an implicit for-all check and instantiation over all $\operatorname{var}(r)$. For each variable assignment corresponding to a ground rule, we require that $B_r^+ \setminus I \neq \emptyset$ or $I \cap (B_r^- \cup H_r) \neq \emptyset$ (Equations (5)–(6)) holds. We use saturation to ensure that a non-ground rule is *satisfied* iff all its variable instantiations are satisfied (Equations (4), (8)) and require satisfiability for all rules (Equations (7), (9)). Saturation carries out the *universal* step covering *all* potential variable instantiations.

Example 3. *We continue Example 2 with the satisfiability check of* r_3*. In Lines 1–2 below, we apply Equation (4) to guess all variable combinations by disjunction, and in Lines 3–6, we apply Equation (5) to handle* $d(X1, X2)$*. In the example, we skip* $d(X1, X3)$ *and* $d(X2, X3)$*.*

```
1 svar_X1(1) | svar_X1(2). svar_X2(1) | svar_X2(2).
2 svar_X3(1) | svar_X3(2).
3 sat_r3 ← svar_X1(1), svar_X2(1), not d(1,1).
```

[2]Saturation (Eiter and Gottlob 1995) is a technique for second-level problems via full disjunctive programs.

Joining $\Pi_{\mathcal{G}}$ and $\Pi_{\mathcal{H}}$

$$r \qquad \text{for every } r \in \mathcal{G}(\Pi_{\mathcal{G}}) \tag{1}$$

Guess Heads of $\Pi_{\mathcal{H}}$

$$\{h'(\mathbf{D})\} \leftarrow \qquad \text{for every } r \in \Pi_{\mathcal{H}}, h(\mathbf{X}) \in H_r, \mathbf{D} \in \mathrm{dom}(\mathbf{X}) \tag{2}$$

$$h(\mathbf{D}) \leftarrow h'(\mathbf{D}) \qquad \text{for every } r \in \Pi_{\mathcal{H}}, h(\mathbf{X}) \in H_r, \mathbf{D} \in \mathrm{dom}(\mathbf{X}) \tag{3}$$

Satisfiability of $\Pi_{\mathcal{H}}$

$$\bigvee_{d \in \mathrm{dom}(\mathbf{x})} \mathrm{svar}_x(d) \leftarrow \qquad \text{for every } r \in \Pi_{\mathcal{H}}, x \in \mathrm{var}(r) \tag{4}$$

$$\mathrm{sat}_r \leftarrow \mathrm{svar}_{x_1}(\mathbf{D}_{\langle x_1 \rangle}), \ldots, \mathrm{svar}_{x_t}(\mathbf{D}_{\langle x_t \rangle}), \neg p(\mathbf{D}) \qquad \text{for every } r \in \Pi_{\mathcal{H}}, p(\mathbf{X}) \in B_r^+, \mathbf{D} \in \mathrm{dom}(\mathbf{X}), \mathbf{X} = \langle x_1, \ldots, x_t \rangle \tag{5}$$

$$\mathrm{sat}_r \leftarrow \mathrm{svar}_{x_1}(\mathbf{D}_{\langle x_1 \rangle}), \ldots, \mathrm{svar}_{x_t}(\mathbf{D}_{\langle x_t \rangle}), p(\mathbf{D}) \qquad \text{for every } r \in \Pi_{\mathcal{H}}, p(\mathbf{X}) \in B_r^- \cup H_r, \mathbf{D} \in \mathrm{dom}(\mathbf{X}), \mathbf{X} = \langle x_1, \ldots, x_t \rangle \tag{6}$$

$$\mathrm{sat} \leftarrow \mathrm{sat}_{r_1}, \ldots, \mathrm{sat}_{r_n} \qquad \text{let } \Pi_{\mathcal{H}} = \{r_1, \ldots, r_n\} \tag{7}$$

$$\mathrm{svar}_x(d) \leftarrow \mathrm{sat} \qquad \text{for every } r \in \Pi_{\mathcal{H}}, x \in \mathrm{var}(r), d \in \mathrm{dom}(x) \tag{8}$$

$$\leftarrow \neg \mathrm{sat} \tag{9}$$

Foundedness of $\Pi_{\mathcal{H}}$

$$\bigvee_{d \in \mathrm{dom}(x)} \mathrm{fvar}_x(d) \leftarrow \qquad \text{for every } r \in \Pi_{\mathcal{H}}, \{h(\mathbf{X})\} = H_r, x \in \mathbf{X} \tag{10}$$

$$1\{\mathrm{bfvar}_y(d_1, \mathbf{D}), \ldots, \mathrm{bfvar}_y(d_k, \mathbf{D})\}1 \leftarrow h'(\mathbf{D}) \qquad \text{for every } r \in \Pi_{\mathcal{H}}, \{h(\mathbf{X})\} = H_r, \mathbf{D} \in \mathrm{dom}(\mathbf{X}), y \in \mathrm{var}(r) \setminus \mathbf{X}, \mathrm{dom}(y) = \{d_1, ..., d_k\} \tag{11}$$

$$\mathrm{fvar}_y(d) \leftarrow \mathrm{bfvar}_y(d, \mathbf{D}), \mathrm{fvar}_{x_1}(d_1), \ldots, \mathrm{fvar}_{x_t}(d_t) \qquad \text{for every } r \in \Pi_{\mathcal{H}}, \{h(\mathbf{X})\} = H_r, \mathbf{X} = \langle x_1, \ldots, x_t \rangle, \mathbf{D} = \langle d_1, \ldots, d_t \rangle, \mathbf{D} \in \mathrm{dom}(\mathbf{X}), y \in \mathrm{var}(r) \setminus \mathbf{X}, d \in \mathrm{dom}(y) \tag{12}$$

$$\mathrm{lit}_p \leftarrow \mathrm{fvar}_{x_1}(\mathbf{D}_{\langle x_1 \rangle}), \ldots, \mathrm{fvar}_{x_t}(\mathbf{D}_{\langle x_t \rangle}), p(\mathbf{D}) \qquad \text{for every } r \in \Pi_{\mathcal{H}}, p(\mathbf{X}) \in B_r^+, \mathbf{D} \in \mathrm{dom}(\mathbf{X}), \mathbf{X} = \langle x_1, \ldots, x_t \rangle \tag{13}$$

$$\mathrm{lit}_p \leftarrow \mathrm{fvar}_{x_1}(\mathbf{D}_{\langle x_1 \rangle}), \ldots, \mathrm{fvar}_{x_t}(\mathbf{D}_{\langle x_t \rangle}), \neg p(\mathbf{D}) \qquad \text{for every } r \in \Pi_{\mathcal{H}}, p(\mathbf{X}) \in B_r^-, \mathbf{D} \in \mathrm{dom}(\mathbf{X}), \mathbf{X} = \langle x_1, \ldots, x_t \rangle \tag{14}$$

$$\mathrm{just}_r \leftarrow \mathrm{lit}_{p_2}, \ldots, \mathrm{lit}_{p_m}, \mathrm{lit}_{p_{m+1}}, \ldots, \mathrm{lit}_{p_n} \qquad \text{for every } r \in \Pi_{\mathcal{H}}, B_r^+ = \{p_2, \ldots, p_m\}, B_r^- = \{p_{m+1}, \ldots, p_n\} \tag{15}$$

$$\mathrm{just}_r \leftarrow \mathrm{fvar}_{x_1}(\mathbf{D}_{\langle x_1 \rangle}), \ldots, \mathrm{fvar}_{x_t}(\mathbf{D}_{\langle x_t \rangle}), \neg h'(\mathbf{D}) \qquad \text{for every } r \in \Pi_{\mathcal{H}}, \{h(\mathbf{X})\} = H_r, \mathbf{D} \in \mathrm{dom}(\mathbf{X}), \mathbf{X} = \langle x_1, \ldots, x_t \rangle \tag{16}$$

$$\mathrm{just} \leftarrow \mathrm{just}_{r_1}, \ldots, \mathrm{just}_{r_n} \qquad \text{let } \Pi_{\mathcal{H}} = \{r_1, \ldots, r_n\} \tag{17}$$

$$\mathrm{fvar}_x(d) \leftarrow \mathrm{just} \qquad \text{for every } r \in \Pi_{\mathcal{H}}, \{h(\mathbf{X})\} = H_r, x \in \mathbf{X}, d \in \mathrm{dom}(x) \tag{18}$$

$$\leftarrow \neg \mathrm{just} \tag{19}$$

Figure 1: FastFound reduction $\mathcal{F}(\Pi_{\mathcal{H}}, \Pi_{\mathcal{G}})$ for a given tight normal program Π that is partitioned into $\Pi_{\mathcal{H}}$ and $\Pi_{\mathcal{G}}$. The reduction uses saturation for the foundedness check, and the resulting grounding size of $\mathcal{F}(\Pi, \emptyset)$ is in $\mathcal{O}\left(|\Pi| \cdot |dom(\Pi)|^{a+1}\right)$.

```
4 sat_r3 ← svar_X1(1), svar_X2(2), not d(1,2).
5 sat_r3 ← svar_X1(2), svar_X2(1), not d(2,1).
6 sat_r3 ← svar_X1(2), svar_X2(2), not d(2,2).
```

The next listing shows the remaining satisfiability rules. In Line 1, we apply Equation (6) to handle $c(X1)$, and in Lines 2–3, we apply Equation (8) to close the saturation loop. Lastly, applying Equations (7), (9) in Line 4 ensures that every answer set satisfies r_3.

```
1 sat_r3 ← svar_X1(1), c(1). sat_r3 ← svar_X1(2), c(2).
2 svar_X1(1) ← sat. svar_X2(1) ← sat. svar_X3(1) ← sat.
3 svar_X1(2) ← sat. svar_X2(2) ← sat. svar_X3(2) ← sat.
4 sat ← sat_r3. ← not sat.
```

3. Foundedness. We must ensure that, for each atom $h(\mathbf{D}) \in I$, there is a rule justifying this atom. A ground normal rule $r \in \mathcal{G}(\Pi)$ justifies $h(\mathbf{D})$, when $\{h(\mathbf{D})\} = H_r$, $B_r^+ \subseteq I$, and $B_r^- \cap I = \emptyset$. While a naive approach fixes $h(\mathbf{D})$, takes a non-ground rule $r \in \Pi$ s.t. $\{h(\mathbf{X})\} = H_r$, and searches through all body instantiations simultaneously, BDG searches in tuples of predicates $\langle h(\mathbf{D}), p(\mathbf{D}_p) \rangle$, where $p(\mathbf{D}_p) \in B_r$. Although BDG's foundedness check is an improvement over the naive approach, it suffers from doubling the arity by searching in tuples $\langle h(\mathbf{D}), p(\mathbf{D}_p) \rangle$, which leads to the *undesirable quadratic increase* in grounding size. This is particularly problematic for practical applications. In fact, it has been open whether one could avoid doubling the arity (Besin, Hecher, and Woltran 2022).

We solve this by introducing the FastFound reduction, which not only decouples body predicates, but *fully decouples* all rule predicates. This is enabled by transitioning from fixing predicates to fixing variable assignments. For each head variable assignment $\mathbf{D} \in \mathrm{dom}(\mathbf{X})$ s.t. $h'(\mathbf{D}) \in I$, we guess one suitable body variable assignment, variable by variable $y \in \mathrm{var}(r) \setminus \mathbf{X}$. Body predicates $p(\mathbf{D}_p) \in B_r$ are instantiated by this variable assignment, and then we check whether $p(\mathbf{D}_p) \in I$. Finally, we implicitly iterate through all head variable assignments with the help of saturation. Therefore, the search for fitting body predicates is only dependent on the variable assignments, which avoids the quadratic overhead in grounding size.

Let $r \in \Pi$ be a non-ground normal rule, let $H_r = \{h(\mathbf{X})\}$ be the rule's head, and let $h(\mathbf{D}) \in \mathrm{dom}(\mathbf{X})$ be an instantiation of the head. Whenever $h'(\mathbf{D}) \notin I$, we can skip the predicate (Equation (16)). Otherwise, a variable instantiation (Equations (11)–(12)) is needed s.t. $B_r^+ \subseteq I$ and $B_r^- \cap I = \emptyset$ (Equations (13)–(14)). Respective conditions must hold for all body predicates in B_r (Equation (15)). Equations (10), (18) perform the universal saturation check, thereby ensuring that all head instantiations (Equation (2)) are justified. Finally, Equation (17) encodes that all rule heads

must be justified, and Equation (19) prevents unfounded interpretations. Note that Equations (2)–(3) ensure unique rule heads. Due to this uniqueness, we can indeed proceed rule by rule and use saturation for the universal step of the head variable assignments, to cover all potential instantiations.

Example 4. *We continue Example 2 by showing the foundedness check of r_3. Line 1 applies Equation (10) and guesses all head-variable combinations. Lines 2–3 apply Equation (11) to obtain the body variable guesses for $X2$. As the body variable guesses are intertwined with a head variable vector, we disentangle them in Lines 4–7 by applying Equation (12). Here, we skip analogous rules for $X3$.*

```
1 fvar_X1(1) | fvar_X1(2).
2 1{bfvar_X2(1,1); bfvar_X2(2,1)}1 ← c'(1).
3 1{bfvar_X2(1,2); bfvar_X2(2,2)}1 ← c'(2).
4 fvar_X2(1) ← bfvar_X2(1,1), fvar_X1(1).
5 fvar_X2(1) ← bfvar_X2(1,2), fvar_X1(2).
6 fvar_X2(2) ← bfvar_X2(2,1), fvar_X1(1).
7 fvar_X2(2) ← bfvar_X2(2,2), fvar_X1(2).
```

Next, we consider the remaining foundedness rules. Lines 1–4 handle $d(X1, X2)$ by applying Equation (13). In a full rewriting, we are required to handle $d(X1, X3)$ and $d(X2, X3)$ as well, which result in the introduction of $\texttt{lit}_{p3}$ and $\texttt{lit}_{p4}$, respectively. When the body holds, the head atom is justified, which is encoded in Line 5 that handles Equation (15). If the head does not hold, we set $\texttt{just}_3$ to true, as in Line 6 handling Equation (16). Line 7 closes the saturation loop by applying Equation (18), and Line 8 ensures that all atoms are founded by applying Equations (17), (19).

```
1 lit_p2 ← fvar_X1(1), fvar_X2(1), d(1,1).
2 lit_p2 ← fvar_X1(1), fvar_X2(2), d(1,2).
3 lit_p2 ← fvar_X1(2), fvar_X2(1), d(2,1).
4 lit_p2 ← fvar_X1(2), fvar_X2(2), d(2,2).
5 just_r3 ← lit_p2, lit_p3, lit_p4.
6 just_r3 ← fvar_X1(1), not c'(1). just_r3 ← fvar_X1(2), not c'(2).
7 fvar_X1(1) ← just. fvar_X1(2) ← just.
8 just ← just_r3. ← not just.
```

Theorem 1 (Correctness)**.** *Let Π be a tight normal program and $\Pi_\mathcal{H} \cup \Pi_\mathcal{G}$ be a partition thereof. Then, the answer sets of $\mathcal{F}(\Pi_\mathcal{H}, \Pi_\mathcal{G})$, given by Figure 1, restricted to HB(Π) bijectively match the answer sets of $\mathcal{G}(\Pi)$.*

Proof. We proceed by proving two directions: (i) every answer set of $\mathcal{G}(\Pi)$ has a corresponding answer set of $\mathcal{F}(\Pi_\mathcal{H}, \Pi_\mathcal{G})$, and the converse that (ii) every answer set of $\mathcal{F}(\Pi_\mathcal{H}, \Pi_\mathcal{G})$ restricted to HB(Π) is an answer set of $\mathcal{G}(\Pi)$. Wlog., we assume that, for any two $r_1, r_2 \in \Pi$, $H_{r_1} \triangle H_{r_2} \neq \emptyset$, $B^+_{r_1} \triangle B^+_{r_2} \neq \emptyset$, or $B^-_{r_1} \triangle B^-_{r_2} \neq \emptyset$, where we ignore variables, to establish that $\mathcal{G}(r_1) \cap \mathcal{G}(r_2) = \emptyset$.

(i) We prove that, for any answer set I of $\mathcal{G}(\Pi)$, there is an answer set I' of $\mathcal{F}(\Pi_\mathcal{H}, \Pi_\mathcal{G})$ s.t. $I' \cap \text{HB}(\Pi) = I$. Let $I' = I \cup \mathbf{SAT} \cup \mathbf{FOUND}$, so the original answer set augmented with parts obtained by satisfiability and foundedness rules. We define the satisfiability part as $\mathbf{SAT} = \{\text{sat}\} \cup \{\text{sat}_r \mid r \in \Pi_\mathcal{H}\} \cup \{\text{svar}_x(d) \mid r \in \Pi_\mathcal{H}, x \in \text{var}(r), d \in \text{dom}(x)\}$, and the foundedness part as $\mathbf{FOUND} = \{\text{just}\} \cup \{\text{just}_r \mid r \in \Pi_\mathcal{H}\} \cup \mathbf{HEADS} \cup \mathbf{LITS} \cup \mathbf{HVAR} \cup \mathbf{BVAR}$, where head variables are $\mathbf{HVAR} = \{\text{fvar}_x(d) \mid r \in \Pi_\mathcal{H}, H_r = \{h(\mathbf{X})\}, x \in \mathbf{X}, d \in \text{dom}(x)\}$. The construction of $\mathbf{HEADS}$, $\mathbf{LITS}$, and $\mathbf{BVAR}$ is detailed in the supplementary material[3].

By this construction, I' is an answer set, as it is the minimal model of $\mathcal{F}(\Pi_\mathcal{H}, \Pi_\mathcal{G})^{I'}$. Due to $I \subseteq I'$, we know that Equation (1) is satisfied. Equations (2)–(8), (10), and (15)–(18) are satisfied as $H_r \subseteq I'$. Equations (9) and (19) are not part of the reduct due to $B^-_r \cap I' \neq \emptyset$. The remaining Equations (11)–(14) are more involved. Equation (11) is satisfied whenever $h'(\mathbf{D}) \notin \mathbf{HEADS}$, or when exactly one $\text{bfvar}_y(d, \mathbf{D}) \in \mathbf{BVAR}_1$. There is precisely one $\text{fvar}_y(d) \in \mathbf{BVAR}_2$ whenever $\text{bfvar}_y(d, \mathbf{D}) \in \mathbf{BVAR}_1$, which satisfies Equation (12). By construction, lit_p are added to $\mathbf{LITS}$ iff the respective body of Equations (13)–(14) holds, which satisfies Equations (13)–(14).

It remains to argue that I' is indeed the minimal model of $\mathcal{F}(\Pi_\mathcal{H}, \Pi_\mathcal{G})^{I'}$. Assume that any $I'' \subset I'$ is a model of $\mathcal{F}(\Pi_\mathcal{H}, \Pi_\mathcal{G})^{I'}$. This implies that $\text{sat} \notin I''$ or $\text{just} \notin I''$, so that I cannot be an answer set of $\mathcal{G}(\Pi)$.

(ii) We prove that, for any answer set I' of $\mathcal{F}(\Pi_\mathcal{H}, \Pi_\mathcal{G})$, $I = I' \cap \text{HB}(\Pi)$ is an answer set of $\mathcal{G}(\Pi)$. Suppose towards a contradiction that I' is an answer set, but I is not. Then, I does not satisfy some rule of $\mathcal{G}(\Pi)$, or some atom $h \in I$ is not founded. Any rule $r \in \mathcal{G}(\Pi_\mathcal{G})$ that is not satisfied is not satisfied in $\mathcal{F}(\Pi_\mathcal{H}, \Pi_\mathcal{G})$ either, due to Equation (1). When a rule $r \in \mathcal{G}(\Pi_\mathcal{H})$ is not satisfied, then $B^+_r \subseteq I$ and $(H_r \cup B^-_r) \cap I = \emptyset$. This means that Equations (5)–(6) cannot justify sat_r, while $\text{sat}_r \in I'$ as I' would not be an answer set otherwise. Since $\text{sat}_r \in I'$ is not founded, I' is not an answer set of $\mathcal{F}(\Pi_\mathcal{H}, \Pi_\mathcal{G})$, which contradicts our assumption.

The other case is that an atom $h \in I$ is not founded. Due to tightness, this translates to h not being justified by any rule. We only need to consider the rules $\mathbf{R}(h) = \{r \in \mathcal{G}(\Pi) \mid \{h\} = H_r\}$. For any $r \in \mathbf{R}(h)$, $B^+_r \not\subseteq I$ or $B^-_r \cap I \neq \emptyset$ holds. As we assume that h is founded in I', it is justified by $\mathcal{F}(\Pi_\mathcal{H}, \Pi_\mathcal{G})$. However, h cannot be justified by any $r \in \mathcal{G}(\Pi_\mathcal{G})$, due to Equation (1). So h must be justified by Equations (2)–(3), and consequently $h' \in I'$ for some $r \in \Pi_\mathcal{H}$ s.t. $\mathcal{G}(r) \cap \mathbf{R}(h) \neq \emptyset$. As I' is an answer set, we require that $\text{just}_r \in I'$, while Equation (16) cannot justify just_r. Since lit_p cannot be justified by Equations (13)–(14) for at least one $p \in B_r$, Equation (15) cannot justify just_r either. Since $\text{just}_r \in I'$ is not founded, I' is not an answer set of $\mathcal{F}(\Pi_\mathcal{H}, \Pi_\mathcal{G})$, which contradicts our assumption. □

Grounding Size Bound. We proceed to bound the grounding size of FastFound. Let Π be a tight normal program, and consider the FastFound reduction $\mathcal{F}(\Pi, \emptyset)$. We denote the maximum head arity by $a_h = \max_{r \in \Pi, h(\mathbf{X}) \in H_r} |\mathbf{X}|$, and the maximum body arity by $a_B = \max_{r \in \Pi, p(\mathbf{X}) \in B_r} |\mathbf{X}|$. Further, let $a = \max\{a_h, a_B\}$ be the maximum arity, $\alpha = \max\{a_h + 1, a_B\}$, and observe that $a + 1 \geq \alpha$.

Theorem 2 (Grounding Size)**.** *Let Π be a tight normal program. Then, the grounding size of $\mathcal{F}(\Pi, \emptyset)$ is in $\mathcal{O}(|\Pi| \cdot |\text{dom}(\Pi)|^\alpha)$.*

[3] https://github.com/alexl4123/newground

Equations (1)–(9) are unchanged
Foundedness Equations (10)–(19) are replaced by:

$$\bigvee_{d\in\mathrm{dom}(x)} \mathrm{fvar}_{h,x}(d) \leftarrow \quad \text{for every } r\in\Pi_{\mathcal{H}}, h(\mathbf{X})\in H_r, x\in\mathbf{X} \tag{20}$$

$$1\{\mathrm{bfvar}_{h,y}(d_1,\mathbf{D}),\ldots,\mathrm{bfvar}_{h,y}(d_k,\mathbf{D})\}1 \leftarrow h'(\mathbf{D}) \quad \text{for every } r\in\Pi_{\mathcal{H}}, h(\mathbf{X})\in H_r, \mathbf{D}\in\mathrm{dom}(\mathbf{X}), y\in\mathrm{var}(r)\setminus\mathbf{X}, \mathrm{dom}(y)=\{d_1,...,d_k\} \tag{21}$$

$$\mathrm{fvar}_{h,y}(d) \leftarrow \mathrm{bfvar}_{h,y}(d,\mathbf{D}),\mathrm{fvar}_{h,x_1}(d_1),\ldots,\mathrm{fvar}_{h,x_t}(d_t) \quad \text{for every } r\in\Pi_{\mathcal{H}}, h(\mathbf{X})\in H_r, \mathbf{X}=\langle x_1,\ldots,x_t\rangle, \mathbf{D}=\langle d_1,\ldots,d_t\rangle, \mathbf{D}\in\mathrm{dom}(\mathbf{X}), y\in\mathrm{var}(r)\setminus\mathbf{X}, d\in\mathrm{dom}(y) \tag{22}$$

$$\mathrm{lit}_{h,p} \leftarrow \mathrm{fvar}_{h,x_1}(\mathbf{D}_{\langle x_1\rangle}),\ldots,\mathrm{fvar}_{h,x_t}(\mathbf{D}_{\langle x_t\rangle}),p(\mathbf{D}) \quad \text{for every } r\in\Pi_{\mathcal{H}}, h\in H_r, p(\mathbf{X})\in B_r^+, \mathbf{D}\in\mathrm{dom}(\mathbf{X}), \mathbf{X}=\langle x_1,\ldots,x_t\rangle \tag{23}$$

$$\mathrm{lit}_{h,p} \leftarrow \mathrm{fvar}_{h,x_1}(\mathbf{D}_{\langle x_1\rangle}),\ldots,\mathrm{fvar}_{h,x_t}(\mathbf{D}_{\langle x_t\rangle}),\neg p(\mathbf{D}) \quad \text{for every } r\in\Pi_{\mathcal{H}}, h\in H_r, p(\mathbf{X})\in(B_r^-\cup(H_r\setminus\{h\})), \mathbf{D}\in\mathrm{dom}(\mathbf{X}), \mathbf{X}=\langle x_1,\ldots,x_t\rangle \tag{24}$$

$$\mathrm{just}_{h_i,r} \leftarrow \mathrm{lit}_{h_i,p_{l+1}},\ldots,\mathrm{lit}_{h_i,p_m},\mathrm{lit}_{h_i,p_{m+1}},\ldots,\mathrm{lit}_{h_i,p_n}, \mathrm{lit}_{h_i,h_1},\ldots,\mathrm{lit}_{h_i,h_{i-1}},\mathrm{lit}_{h_i,h_{i+1}},\ldots,\mathrm{lit}_{h_i,h_l} \quad \text{for every } r\in\Pi_{\mathcal{H}}, H_r=\{h_1,\ldots,h_i,\ldots,h_l\}, B_r^+=\{p_{l+1},\ldots,p_m\}, B_r^-=\{p_{m+1},\ldots,p_n\} \tag{25}$$

$$\mathrm{just}_{h,r} \leftarrow \mathrm{fvar}_{h,x_1}(\mathbf{D}_{\langle x_1\rangle}),\ldots,\mathrm{fvar}_{h,x_t}(\mathbf{D}_{\langle x_t\rangle}),\neg h'(\mathbf{D}) \quad \text{for every } r\in\Pi_{\mathcal{H}}, h(\mathbf{X})\in H_r, \mathbf{D}\in\mathrm{dom}(\mathbf{X}), \mathbf{X}=\langle x_1,\ldots,x_t\rangle \tag{26}$$

$$\mathrm{just} \leftarrow \mathrm{just}_{h_{1_1},r_1},\ldots,\mathrm{just}_{h_{l_1},r_1},\ldots,\mathrm{just}_{h_{1_n},r_n},\ldots,\mathrm{just}_{h_{l_n},r_n} \quad \text{let } \Pi_{\mathcal{H}}=\{r_1,\ldots,r_n\}, \forall r_i\in\Pi_{\mathcal{H}}: H_{r_i}=\{h_{1_i},\ldots,h_{l_i}\} \tag{27}$$

$$\mathrm{fvar}_{h,x}(d) \leftarrow \mathrm{just} \quad \text{for every } r\in\Pi_{\mathcal{H}}, h(\mathbf{X})\in H_r, x\in\mathbf{X}, d\in\mathrm{dom}(x) \tag{28}$$

$$\leftarrow \neg\mathrm{just} \tag{29}$$

Figure 2: Adapted FastFound reduction $\mathcal{F}_H(\Pi_{\mathcal{H}},\Pi_{\mathcal{G}})$ for a given tight HCF program Π that is partitioned into $\Pi_{\mathcal{H}}$ and $\Pi_{\mathcal{G}}$. The resulting grounding size of $\mathcal{F}_H(\Pi,\emptyset)$ is in $\mathcal{O}\left(h\cdot|\Pi|\cdot|dom(\Pi)|^{a+1}\right)$.

Proof. Equations (9) and (19) have a grounding size that is in $\mathcal{O}(1)$. Equations (7), (15), and (17) have a grounding size that is in $\mathcal{O}(|\Pi|)$. Equations (4), (8), (10), and (18) have a grounding size that is in $\mathcal{O}(|\Pi|\cdot|\,\mathrm{dom}(\Pi)|)$. Equations (2), (3), (5), (6), (13), (14), and (16) have a grounding size that is in $\mathcal{O}(|\Pi|\cdot|\,\mathrm{dom}(\Pi)|^a)$. Equations (11) and 12) have a grounding size that is in $\mathcal{O}(|\Pi|\cdot|\,\mathrm{dom}(\Pi)|^{a_h+1})$.

So, $\mathcal{O}\left(|\Pi|\cdot|\,\mathrm{dom}(\Pi)|^{\max\{a_h+1,a\}}\right)$ is the grounding size. As $\max\{a_h{+}1,a\}=\max\{a_h{+}1,\max\{a_h,a_B\}\}=\max\{a_h{+}1,a_B\}=\alpha$, we obtain $\mathcal{O}\left(|\Pi|\cdot|\,\mathrm{dom}(\Pi)|^{\alpha}\right)$. □

As $a+1\geq\alpha$, we write $\mathcal{O}\left(|\Pi|\cdot|\,\mathrm{dom}(\Pi)|^{a+1}\right)$ for brevity.

Example 5. *Recall r_3 of Example 1. Observe that $a_h=1$, $a_b=2$, and therefore $\alpha=2$. Evidently, FastFound's grounding size of r_3 is $\approx|\,\mathrm{dom}(\Pi)|^2$, compared to $\approx|\,\mathrm{dom}(\Pi)|^3$ for both BDG and bottom-up grounding. Letting $\Pi_{\mathcal{G}}=\{r_1\}$ and $\Pi_{\mathcal{H}}=\{r_2,r_3\}$, the grounding size of $\mathcal{F}(\Pi_{\mathcal{H}},\Pi_{\mathcal{G}})$ is in $\approx|\,\mathrm{dom}(\Pi)|^2$.*

3.3 FastFound for Tight HCF Programs

In this section, we extend the FastFound reduction from tight normal programs to tight HCF programs. Our method is inspired by the shifting technique (Gelfond et al. 1991) for ground programs, which transforms HCF rules into normal rules. Shifting splits an HCF rule r, with l-many head predicates, into l-many rules. Each fresh rule r_i has one head predicate $\{h_i(\mathbf{X}_i)\}=H_{r_i}$, where the remaining $H_r\setminus\{h_i(\mathbf{X}_i)\}$ head predicates are moved to the negative body $B_{r_i}^-$. Shifting preserves answer sets for HCF programs (Dechter 1994).

In Figure 2, the FastFound reduction for HCF programs is shown. On a high level, the tight HCF reduction is similar to the tight normal reduction, as we have three parts: *atom guessing*, *proving satisfiability*, and *proving foundedness*. However, due to the nature of atom guessing and the definition of satisfiability, we do not need to apply shifting to Equations (1)–(9). For satisfiability, this stems from $I\cap(B_r^-\cup H_r)\neq\emptyset$, as we handle rule heads with $|H_r|>1$ in Equation (6) as well.

Therefore, we are only required to integrate shifting into the foundedness part, which translates to a *partial application of shifting*, where the rules for foundedness are adapted by Equations (20)–(29). Intuitively, the new foundedness rules incorporate the shifting idea into FastFound (for non-ground programs) by indexing predicates with the head predicates.

Equations (20)–(22) and (28) index the respective predicates on a per-head predicate basis $h(\mathbf{X})\in H_r$, while Equation (29) stays. Equations (23) and (26) treat $h(\mathbf{X})$ from the head H_r, while Equation (24) considers all $H_r\setminus\{h(\mathbf{X})\}$ as a negative body predicate. Lastly, Equation (25) requires all $H_r\setminus\{h(\mathbf{X})\}$ predicates to hold, and Equation (27) is adapted to check all rules and heads.

Theorem 3 (Correctness)**.** *Let Π be a tight HCF program and $\Pi_{\mathcal{H}}\cup\Pi_{\mathcal{G}}$ be a partition thereof. Then, the answer sets of $\mathcal{F}_H(\Pi_{\mathcal{H}},\Pi_{\mathcal{G}})$, given by Figure 2, restricted to HB(Π) bijectively match the answer sets of $\mathcal{G}(\Pi)$.*

Proof (Sketch). We proceed analogously to the proof of Theorem 1. Here we present the idea, while a more detailed proof is given in the supplementary material.

(i): Let I be an answer set of $\mathcal{G}(\Pi)$, from which we construct $I'=I\cup\mathbf{SAT}\cup\mathbf{FOUND}$. $\mathbf{SAT}$ is constructed in the same way as in the proof of Theorem 1, whereas the construction of $\mathbf{FOUND}$ needs adaptation. We need to pay special attention to include the proper indexing and to the definition of $\mathbf{LITS}$. For the construction, we then show that I' is indeed an answer set of $\mathcal{F}_H(\Pi_{\mathcal{H}},\Pi_{\mathcal{G}})$ by arguing that

I' is a model as well as a minimal model of $\mathcal{F}_H(\Pi_\mathcal{H}, \Pi_\mathcal{G})^{I'}$, as I would not be an answer set of $\mathcal{G}(\Pi)$ otherwise.

(ii): Similarly to the proof of Theorem 1, let I' be an answer set of $\mathcal{F}_H(\Pi_\mathcal{H}, \Pi_\mathcal{G})$ and $I = I' \cap \text{HB}(\Pi)$. As the shifted program Π_S is normal, it suffices to show that all rules of Π_S are satisfied and all atoms in I are founded. Towards a contradiction, we assume that I is not an answer set of Π_S, which means that some rule of Π_S is not satisfied or some atom in I is not founded. However, either condition contradicts our assumption that I' is an answer set of $\mathcal{F}_H(\Pi_\mathcal{H}, \Pi_\mathcal{G})$. □

We proceed by stating our grounding size bound of FastFound for HCF programs. Let the maximum head arity be a_h, the maximum body arity be a_B, the maximum arity be a, and α be defined as in the tight normal case. Further, let $h = \max_{r \in \Pi} |H_r|$ be the maximum number of head predicates.

Theorem 4 (Grounding Size)**.** *Let Π be a tight HCF program. Then, the grounding size of $\mathcal{F}_H(\Pi, \emptyset)$ is in $\mathcal{O}\left(h \cdot |\Pi| \cdot |\operatorname{dom}(\Pi)|^\alpha\right)$.*

Proof (Sketch). We obtain the same arities as for $\mathcal{F}$. However, Equations (20)–(26) and (28) can occur h-many times. □

3.4 FastFound for Normal Programs

For a normal program Π, we must additionally prevent cyclic derivations, where FastFound relies on level mappings (Janhunen 2006). We briefly sketch their integration, saying that an SCC S is non-tight whenever $|S| \geq 2$. Level mappings are required whenever there is a non-tight SCC S with some predicate $p \in S$ s.t. $\text{RULES}(p) \cap \Pi_\mathcal{H} \neq \emptyset$. We construct level mappings for any two atoms $p_1(\mathbf{D}_1), p_2(\mathbf{D}_2) \in S$, where we guess whether $p_1(\mathbf{D}_1)$ is derived before $p_2(\mathbf{D}_2)$, denoted by $[p_1(\mathbf{D}_1) \prec p_2(\mathbf{D}_2)]$, or the contrary $[p_2(\mathbf{D}_2) \prec p_1(\mathbf{D}_1)]$. Additionally, we prevent non-transitive derivations, and adapt the FastFound reduction to require that, for each $h \in H_r$ and $p \in B_r^+$, also $[p(\mathbf{D}_p) \prec h(\mathbf{D}_h)]$ must hold. For a normal program Π partitioned into $\Pi_\mathcal{H}$ and $\Pi_\mathcal{G}$, we write $\mathcal{F}_{lv}(\Pi_\mathcal{H}, \Pi_\mathcal{G})$ for FastFound with level mappings.

Theorem 5. *Let Π be a normal program and $\Pi_\mathcal{H} \cup \Pi_\mathcal{G}$ be a partition thereof. Then, the answer sets of $\mathcal{F}_{lv}(\Pi_\mathcal{H}, \Pi_\mathcal{G})$ restricted to HB(Π) match the answer sets of $\mathcal{G}(\Pi)$.*

Proof (Sketch). On a high level, we proceed as in the proof of Theorem 1. In the construction of **(i)**, we must additionally include the $[p(\mathbf{D}_p) \prec h(\mathbf{D}_h)]$ predicates and argue that no cyclic derivations can occur. In **(ii)**, we proceed with a proof by contradiction and the additional case of cyclic derivations for I not being an answer set. See the supplementary material for details. □

We obtain the following result on grounding size. Let Π be a normal program, $\mathcal{D}$ be the (positive) dependency graph of Π, $\text{SCC}(\Pi)$ be the SCCs of $\mathcal{D}$, and $\text{SCC}_{nt}(\Pi) = \{S \in \text{SCC}(\Pi) \mid |S| > 1\}$ be the non-trivial SCCs. Further, let α be defined as previously, $\beta = 3 \cdot \max_{S \in \text{SCC}_{nt}(\Pi), p \in S} |p|$, and $\gamma = \max\{\alpha, \beta\}$.

Theorem 6. *Let Π be a normal program. Then, the grounding size of $\mathcal{F}_{lv}(\Pi, \emptyset)$ is in $\mathcal{O}\left(|\Pi| \cdot |dom(\Pi)|^\gamma\right)$.*

Proof (Sketch). We proceed analogously to Theorem 2. Details are given in the supplementary material. □

As $3 \cdot a \geq \gamma$, we write $\mathcal{O}\left(|\Pi| \cdot |dom(\Pi)|^{3 \cdot a}\right)$. FastFound can also be extended to HCF programs by incorporating level mappings into Figure 2.

4 Interoperability With Disjunctions

We define conditions for interoperability of FastFound when using non-ground disjunctive programs. So far, no results for interoperability of BDG with non-ground disjunctive programs were known.

Recall that the computational complexity of the answer set existence problem under bounded predicate arities for non-ground HCF/normal programs is Σ_2^P-complete. However, the complexity of the answer set existence problem increases to Σ_3^P-completeness for non-ground full disjunctive programs (Eiter et al. 2007). Therefore, an adaptation of the FastFound reduction that is applicable to non-ground full disjunctive programs in general is highly unlikely. Still, we are able to define conditions under which an application to parts of a non-ground disjunctive program is possible.

Example 6. *Consider the non-ground disjunctive program Π displayed in the next listing, where H and K are arbitrary (integer) constants. The rules in Lines 1–2 define a toy saturation example over predicates a, z, and disj. As input, we assume a graph instance defined by predicate e (Line 1). Lines 3–4, which we denote as r_3 and r_4, respectively, guess subgraphs defined by predicate f, where predicate d derives vertices of cliques of size greater or equal to 3.*

```
1 a(1)|..|a(K). {z(1);..;z(K)}. e(1,2). .. e(H,K).
2 disj ← a(X), not z(X). a(1) ← disj. .. a(K) ← disj.
3 {f(X,Y)} ← e(X,Y), a(X).
4 d(X1) ← f(X1,X2), f(X1,X3), f(X2,X3).
```

We show the dependency graph of Π in Figure 3. As predicate a is part of a disjunctive rule and a non-tight SCC S_{disj}, it is not clear how FastFound could be applied to Π. However, the application of FastFound to r_4 is beneficial and correct, due to r_4's effective independence of S_{disj}.

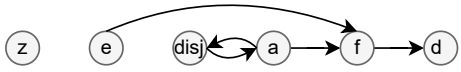


Figure 3: Dependency graph of Π.

By moving from intuition to computation, we start by defining the reduction $\mathcal{F}_i$, a combination of $\mathcal{F}_H$ with $\mathcal{F}_{lv}$. Let Π be a full disjunctive program, $\Pi_\mathcal{H} \cup \Pi_\mathcal{G} = \Pi$ be a partition, and $\Pi_{\mathcal{F}_H} \cup \Pi_{\mathcal{F}_{\text{lv}}} = \Pi_\mathcal{H}$ be another partition. In the latter partition, $\Pi_{\mathcal{F}_{\text{lv}}}$ is the set of rules occurring in non-tight normal SCCs, $\Pi_{\mathcal{F}_{\text{lv}}} = \{r \in \Pi_\mathcal{H} \mid \{h\} = H_r, \forall p \in \text{SCC}(\Pi, h). \forall r' \in \text{RULES}(p) : |H_{r'}| \leq 1\}$, and $\Pi_{\mathcal{F}_H} = \Pi_\mathcal{H} \setminus \Pi_{\mathcal{F}_{\text{lv}}}$ is the rest. In our theorem, we require conditions on $\Pi_\mathcal{H}$. Then, we define $\mathcal{F}_i(\Pi_\mathcal{H}, \Pi_\mathcal{G})$ by $\mathcal{F}_i(\Pi_\mathcal{H}, \Pi_\mathcal{G}) = \mathcal{F}_{\text{lv}}(\Pi_{\mathcal{F}_{\text{lv}}}, \Pi_\mathcal{G}) \cup \mathcal{F}_H(\Pi_{\mathcal{F}_H}, \emptyset)$.

Theorem 7. *(Interoperability for Full Disjunctive ASP) Let Π be a disjunctive program and $\Pi_\mathcal{H} \cup \Pi_\mathcal{G}$ be a partition*

Scenario	Instances	gringo			idlv			BDG			FastFound		
		S	M	T	S	M	T	S	M	T	S	M	T
Total-SUM	240	23	182	35	29	182	29	54	153	33	**75**	131	34
CN3	40	7	33	0	7	33	0	**35**	5	0	**35**	5	0
CC3	40	5	35	0	5	35	0	5	35	0	**17**	23	0
CC4	40	1	39	0	1	39	0	3	37	0	**13**	27	0
CP3	40	**5**	35	0	**5**	35	0	4	36	0	3	37	0
CP4	40	1	39	0	1	39	0	1	39	0	**2**	38	0
LS($N \times N$)	40	4	1	35	**10**	1	29	6	1	33	5	1	34

Table 1: Numbers of solved instances (S), memouts (M), and timeouts (T) for each scenario, comparing gringo, idlv, BDG, and FastFound. Timeouts occur after 1800s, and memouts occur if the RAM usage exceeds 10GB.

s.t., for each $r \in \Pi_{\mathcal{H}}$ and $p \in H_r$, at least one of these two conditions holds:

1. $|\operatorname{SCC}(\Pi, p)| = 1$, *or*
2. $S = \operatorname{SCC}(\Pi, p)$ *and* $\forall q \in S. \forall r \in$ *RULES*(q)*:* $|H_r| \leq 1$.

Then, the answer sets of $\mathcal{F}_i(\Pi_{\mathcal{H}}, \Pi_{\mathcal{G}})$ restricted to HB(Π) bijectively match the answer sets of $\mathcal{G}(\Pi)$.

Proof (Sketch). The outline of the proof is similar to the proof of Theorem 1, while details are given in the supplementary material. We prove both sides of the bijection: (i) every answer set of $\mathcal{G}(\Pi)$ has a corresponding answer set of $\mathcal{F}_i(\Pi_{\mathcal{H}}, \Pi_{\mathcal{G}})$, and (ii) every answer set of $\mathcal{F}_i(\Pi_{\mathcal{H}}, \Pi_{\mathcal{G}})$ restricted to HB(Π) is an answer set of $\mathcal{G}(\Pi)$.

(i): Let I be an answer set of $\mathcal{G}(\Pi)$, from which we construct $I' = I \cup \mathbf{SAT} \cup \mathbf{FOUND}$. Special care must be taken for the construction of **FOUND**, as due to our conditions, any rule $r \in \Pi_{\mathcal{H}}$ can be tight normal, normal, or HCF, where either $\mathcal{F}_H$ or $\mathcal{F}_{lv}$ is selectively applied. It remains to show that I' is indeed an answer set of $\mathcal{F}_i(\Pi_{\mathcal{H}}, \Pi_{\mathcal{G}})$ by arguing that I' is a model as well as a minimal model of $\mathcal{F}_i(\Pi_{\mathcal{H}}, \Pi_{\mathcal{G}})^{I'}$.

(ii): We show that, for any answer set I' of $\mathcal{F}_i(\Pi_{\mathcal{H}}, \Pi_{\mathcal{G}})$, $I = I' \cap \text{HB}(\Pi)$ is an answer set of $\mathcal{G}(\Pi)$. As Π is a full disjunctive program, we cannot directly reuse the satisfiability and foundedness arguments from the prior proofs to establish that I is a minimal model of $\mathcal{G}(\Pi)^I$. Towards a contradiction, we assume that I is not an answer set of $\mathcal{G}(\Pi)$, where we consider the two cases that I is not a model or no minimal model of $\mathcal{G}(\Pi)^I$. The first case is analogous to the satisfiability part in the proof of Theorem 1. In the second case, we pay special attention to the disjunctive rules of Π. □

Note that the conditions for interoperability of FastFound carry over to BDG (Besin, Hecher, and Woltran 2022).

Example 7. *We illustrate the application of Theorem 7 on the program Π from Example 6. Let $\Pi_{\mathcal{H}} = \{r_4\}$ and $\Pi_{\mathcal{G}} = \Pi \setminus \Pi_{\mathcal{H}}$. As $|\operatorname{SCC}(\Pi, d)| = 1$, FastFound can be used to ground $\mathcal{F}_i(\Pi_{\mathcal{H}}, \Pi_{\mathcal{G}})$. Further, observe that $|\operatorname{SCC}(\Pi, disj)| = |\operatorname{SCC}(\Pi, a)| = 2$, while it is 1 for all other (head) predicates occurring in $\Pi_{\mathcal{G}}$.*

5 Experiments

We demonstrate the viability of FastFound by conducting experiments. Therefore, we implemented the tight FastFound reduction in our prototype newground, which is written in Python (version 3.12.1). We restrict our analysis to comparing grounding-heavy scenarios.

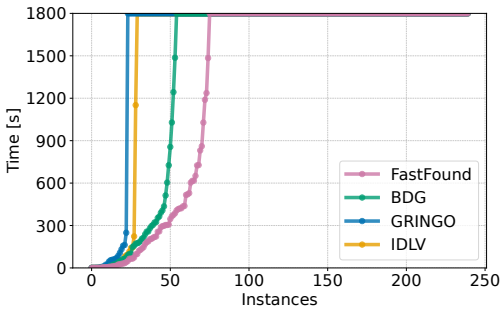
Scenario	gringo	idlv	BDG	FastFound
CN3	3	3	2	2
CC3	3	3	3	2
CC4	4	4	3	2
CP3	3	3	3	3
CP4	4	4	4	3
LS($N \times N$)	$4N-1$	$\approx N+1$	5	4

Table 2: Asymptotic grounding sizes of the instances for gringo, idlv, BDG, and FastFound, where table entries show the exponent c of $\mathcal{O}\left(|\Pi| \cdot |\operatorname{dom}(\Pi)|^c\right)$.

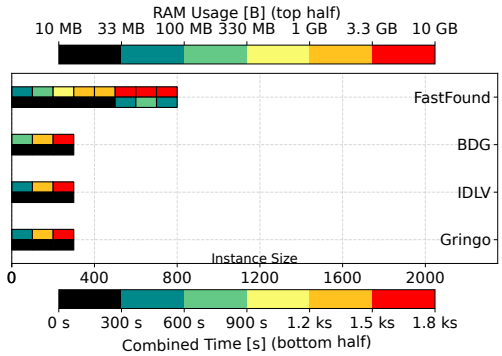
Benchmark System. We compare gringo (version 5.7.1), idlv (version 1.1.6), BDG (with newground version 3.0.0), and FastFound (with newground version 3.0.0). As ASP solver, we run clingo (version 5.7.1) with clasp (version 3.3.10). For both BDG and FastFound, we take gringo as the standard grounder. Our benchmark system has *225GB* RAM and an *AMD Opteron 6272* CPU with 16 cores, operated by *Debian 10* OS (kernel 4.19.0-16-amd64).

Benchmark Setup. We measure *grounding time*, *grounding size*, *ram usage*, and *combined time* (grounding and solving). The grounding performance was evaluated in a separate run. We consider instances as a *TIMEOUT* whenever a system takes more than *1800s* (grounding or combined time), and a *MEMOUT* in case RAM usage or grounding size exceeds *10 GB*. We set a fixed seed for clingo (*11904657*) and ensured that each experiment is run on a separate physical processor, to obtain near-deterministic results.

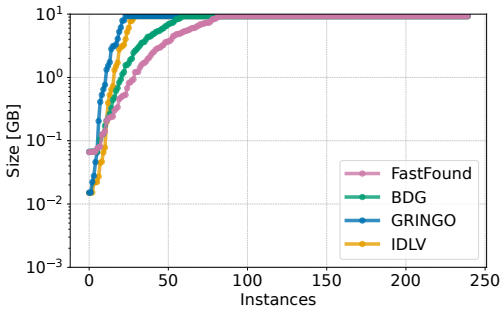
Benchmarks Scenarios. We take grounding-heavy benchmarks adapted from BDG (Besin, Hecher, and Woltran 2022) and hybrid grounding experiments (Beiser et al. 2024), which we adapt for normal rules. These scenarios take a graph as input, where we generate complete graphs ranging from instance size 50 to 2000 with step increases of 50. In more detail, we benchmark the scenarios of constrained cliques (CN3), counting cliques (CC3, CC4), and counting paths (CP3, CP4). In addition, we encode latin squares (LS) with long rules, where latin squares are of the size $N \times N$, ranging



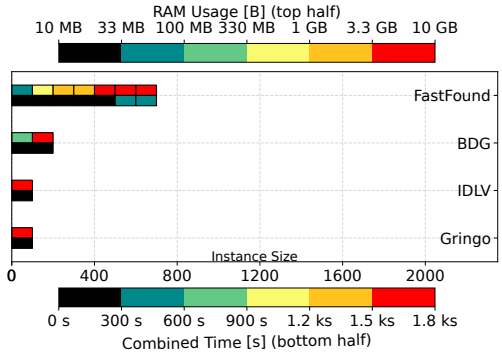
(a) Combined time [s] cactus plot.



(b) Solving profile for the CC3 scenario.



(c) Max RAM usage [GB] cactus plot.



(d) Solving profile for the CC4 scenario.

Figure 4: Cactus plots (Figures 4a and 4c) and solving profiles (Figures 4b and 4d), comparing `gringo`, `idlv`, BDG, and FastFound.

from $N = 1$ to $N = 40$. For each scenario, $\Pi_{\mathcal{H}}$ contains a single grounding-heavy rule, whereas $\Pi_{\mathcal{G}} = \Pi \setminus \Pi_{\mathcal{H}}$. Table 2 shows asymptotic grounding sizes, where we assume that structural rewritings are used for `idlv` (Calimeri et al. 2018; Bichler, Morak, and Woltran 2016). Also note that, for these scenarios, `ProASP` (Dodaro, Mazzotta, and Ricca 2024) is not applicable due to syntax restrictions.

Hypotheses. We expect to observe the following behavior:

H1 FastFound is able to solve previously ungroundable instances on grounding-heavy scenarios.

H2 If asymptotic grounding sizes of FastFound, BDG, and bottom-up solvers match, bottom-up systems solve more instances than BDG, which still outperforms FastFound.

Experiment Results and Discussion. We show overall results, cactus plots, and solving profiles in Table 1 and Figure 4. Overall, FastFound solves the most instances (75), which is followed by BDG (54), `idlv` (29), and `gringo` (23). Figures 4b and 4d show that FastFound is able to solve larger instances of the counting cliques scenarios CC3 and CC4 than `gringo`, `idlv`, and BDG. Further, the cactus plot in Figure 4c exhibits a reduced increase in RAM usage of FastFound in comparison to the other grounders. Relating this to Figure 4a, we observe that excessive RAM usage is indeed the main reason for solving failures, confirming H1.

Regarding H2, let us inspect the scenario CP3, where the asymptotic grounding sizes of `gringo`, `idlv`, BDG, and FastFound match. Here, `gringo` and `idlv` solve 5 instances, whereas BDG and FastFound solve 4 or 3 instances, respectively. Accordingly, we can confirm H2 as well.

The main conclusion of H1 and H2 is that only an informed usage of BDG or FastFound is beneficial. Ideally, the decision which rules to include in $\Pi_{\mathcal{H}}$ should be made automatically by a decision heuristic. We expect this to be embeddable into standard grounders.

6 Discussion and Conclusion

Alternative grounding techniques show promising results. However, using some of them, e.g., BDG, still remains a challenge. In this paper, we introduce FastFound: an alternative BDG foundedness check that avoids the quadratic increase of grounding size for normal rules. We extend the basic FastFound reduction from tight normal programs to HCF programs, and then further to the non-tight case. Additionally, we obtain novel interoperability results for FastFound on full disjunctive programs. We implemented FastFound in our `newground` prototype, which shows promising results on grounding-heavy benchmarks. Future research should investigate the challenging issue of programs with high-arity predicates and a symbiosis with lazy grounding.

Acknowledgements

This research was supported by Frequentis, the Austrian Science Fund (FWF), grants 10.55776/COE12 and

10.55776/J4656, and the Dieberger-Peter Skalicky Stipend.

References

Arias, J.; Carro, M.; Salazar, E.; Marple, K.; and Gupta, G. 2018. Constraint answer set programming without grounding. *TPLP* 18(3-4):337–354.

Banbara, M.; Kaufmann, B.; Ostrowski, M.; and Schaub, T. 2017. *Clingcon* : The next generation. *TPLP* 17(4):408–461.

Beiser, A.; Hecher, M.; Unalan, K.; and Woltran, S. 2024. Bypassing the ASP Bottleneck: Hybrid Grounding by Splitting and Rewriting. In *IJCAI24*, 3250–3258.

Besin, V.; Hecher, M.; and Woltran, S. 2022. Body-Decoupled Grounding via Solving: A Novel Approach on the ASP Bottleneck. In *IJCAI22*, 2546–2552.

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 *ECAI23*, volume 372 of *FAIA*, 247–254.

Bichler, M.; Morak, M.; and Woltran, S. 2016. lpopt: A Rule Optimization Tool for Answer Set Programming. In *LOPSTR16*, volume 10184 of *LNCS*, 114–130.

Bomanson, J.; Janhunen, T.; and Weinzierl, A. 2019. Enhancing Lazy Grounding with Lazy Normalization in Answer-Set Programming. In *AAAI19*, volume 33, 2694–2702.

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

Calimeri, F.; Fuscà, D.; Perri, S.; and Zangari, J. 2018. Optimizing Answer Set Computation via Heuristic-Based Decomposition. *PADL18* 10702:135–151.

Calimeri, F.; Faber, W.; Gebser, M.; Ianni, G.; Kaminski, R.; Krennwallner, T.; Leone, N.; Maratea, M.; Ricca, F.; and Schaub, T. 2020. ASP-Core-2 Input Language Format. *TPLP* 20(2):294–309.

Cuteri, B.; Dodaro, C.; Ricca, F.; and Schüller, P. 2019. Partial Compilation of ASP Programs. *TPLP* 19(5):857–873.

Dantsin, E.; Eiter, T.; and Gottlob, G. 2001. Complexity and expressive power of logic programming. *ACM Comput. Surv.* 33(3):374–425.

Dechter, R. 1994. Propositional Semantics for Disjunctive Logic Programs. *Ann. Math. Artif. Intell.* 12(1-2):53–87.

Dodaro, C.; Mazzotta, G.; and Ricca, F. 2024. Blending Grounding and Compilation for Efficient ASP Solving. In *KR24*, 317–328.

Eiter, T., and Gottlob, G. 1995. On the computational cost of disjunctive logic programming: Propositional case. *Ann. Math. Artif. Intell.* 15(3):289–323.

Eiter, T.; Faber, W.; Fink, M.; and Woltran, S. 2007. Complexity results for answer set programming with bounded predicate arities and implications. *Ann. Math. Artif. Intell.* 51(2):123–165.

Erdem, E.; Gelfond, M.; and Leone, N. 2016. Applications of Answer Set Programming. *AI Magazine* 37(3):53–68.

Falkner, A.; Friedrich, G.; Schekotihin, K.; Taupe, R.; and Teppan, E. C. 2018. Industrial Applications of Answer Set Programming. *Künstl. Intell.* 32(2):165–176.

Gebser, M.; Harrison, A.; Kaminski, R.; Lifschitz, V.; and Schaub, T. 2015. Abstract gringo. *TPLP* 15(4):449–463.

Gebser, M.; Kaminski, R.; Kaufmann, B.; Ostrowski, M.; Schaub, T.; and Wanko, P. 2016. Theory Solving Made Easy with Clingo 5. *ICLP16* 52:1–15.

Gebser, M.; Leone, N.; Maratea, M.; Perri, S.; Ricca, F.; and Schaub, T. 2018. Evaluation Techniques and Systems for Answer Set Programming: a Survey. In *IJCAI18*, 5450–5456.

Gebser, M.; Kaminski, R.; and Schaub, T. 2015. Grounding Recursive Aggregates: Preliminary Report. *GTTV15*.

Gelfond, M., and Leone, N. 2002. Logic programming and knowledge representation—The A-Prolog perspective. *Artif. Intell.* 138(1):3–38.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New. Gener. Comput.* 9(3):365–385.

Gelfond, M.; Lifschitz, V.; Przymusinska, H.; and Truszczynski, M. 1991. Disjunctive Defaults. In *KR91*, 230–237.

Janhunen, T. 2006. Some (in)translatability results for normal logic programs and propositional theories. *JANCL* 16(1-2):35–86.

Kaminski, R., and Schaub, T. 2023. On the Foundations of Grounding in Answer Set Programming. *TPLP* 23(6):1138–1197.

Leone, N.; Pfeifer, G.; Faber, W.; Eiter, T.; Gottlob, G.; Perri, S.; and Scarcello, F. 2006. The DLV system for knowledge representation and reasoning. *TOCL* 7(3):499–562.

Leutgeb, L., and Weinzierl, A. 2018. Techniques for Efficient Lazy-Grounding ASP Solving. *DECLARE18* 10997:132–148.

Lierler, Y., and Robbins, J. 2021. DualGrounder: Lazy Instantiation via Clingo Multi-shot Framework. *JELIA21* 12678:435–441.

Lin, F., and Zhao, J. 2003. On tight logic programs and yet another translation from normal logic programs to propositional logic. In *IJCAI03*, 853–858.

Liu, G.; Janhunen, T.; and Niemela, I. 2012. Answer Set Programming via Mixed Integer Programming. In *KR12*, 32–42.

Mazzotta, G.; Ricca, F.; and Dodaro, C. 2022. Compilation of Aggregates in ASP Systems. In *AAAI22*, volume 36, 5834–5841.

Weinzierl, A. 2017. Blending Lazy-Grounding and CDNL Search for Answer-Set Solving. In *LPNMR17*, volume 10377 of *LNCS*, 191–204.