TU WIEN Informatics

Body-Decoupled Grounding in Normal Answer Set Programs

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Software & Information Engineering

eingereicht von

Kaan Unalan
Matrikelnummer 11907099

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Woltran

Mitwirkung: Dipl.-Ing. Dr.techn. Markus Hecher, BSc

Wien, 1. Juli 2022

Kaan Unalan	Stefan Woltran

TU WIEN Informatics

Body-Decoupled Grounding in Normal Answer Set Programs

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software & Information Engineering

by

Kaan Unalan
Registration Number 11907099

to the Faculty of Informatics
at the TU Wien

Advisor: Univ.Prof. Dipl.-Ing. Dr.techn. Stefan Woltran

Assistance: Dipl.-Ing. Dr.techn. Markus Hecher, BSc

Vienna, 1st July, 2022

Kaan Unalan	Stefan Woltran

Erklärung zur Verfassung der Arbeit

Kaan Unalan

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 1. Juli 2022

Kaan Unalan

Acknowledgements

This work is a result of collective efforts of a large team. In particular, I thank to Stefan Woltran for introducing me this interesting topic and giving me the opportunity to write my thesis about it. I would also like to thank Markus Hecher for his valuable guidance, support and helpful answers to all my questions. Furthermore, I would like to thank my friends and everyone who contributed to this thesis, perhaps without even knowing it. Finally, I express my sincerest gratitude to my family. Without their support, I would not be the person that I am today so this work could not be finished (or even started) at all.

Kurzfassung

Answer Set Programming (ASP) ist ein Programmierparadigma, das in vielen industriellen und akademischen Projekten erfolgreich angewendet wird, um vor allem schwierige Probleme zu lösen. Einer der wichtigsten Gründe dafür ist die Existenz effizienter ASP-Systeme, die meistens in zwei Schritten zur Lösung kommen: Grounding (Grundieren) und Solving (Lösen). Trotzdem bereitet der erstere, nämlich Grounding, bei dem Variablen durch Konstanten ersetzt werden, immer noch große Schwierigkeiten, da dieser Schritt das Programm wesentlich vergrößern kann. Das grundierte Programm ist dann so groß, dass das Solving zu lange dauert.

Body-Decoupled Grounding (körperentkoppeltes Grundieren) ist ein neuer Ansatz, der das Problem des ineffizienten Groundings zu überwinden versucht. Dazu werden die Prädikate in Regelkörpern entkoppelt und deren Auswertung an den Solver delegiert, was anhand einer Reduktion von einem nicht grundierten normalen Programm auf ein grundiertes disjunktives Programm bewerkstelligt wird. Die Komplexität des Verfahrens ist nur in Abhängigkeit von der maximalen Stelligkeit der Prädikate im Programm exponentiell, daher polynomiell, wenn die maximale Stelligkeit durch eine Konstante beschränkt wird.

Diese Arbeit befasst sich mit Body-Decoupled Grounding für normale Programme. Für die Reduktion werden zwei verschiedene Varianten präsentiert, um die Fundiertheit von Interpretationen zu gewährleisten, insbesondere wenn die Programme Zyklen enthalten. Das praktische Potenzial der beiden Vorschläge wird mittels eines Experiments vorgelegt. Die Implementierungen der beiden Varianten werden miteinander, mit einem modernen ASP-System und einer weiteren Implementierung, die in zyklenfreien Programmen eingesetzt wird, hinsichtlich der Größe des grundierten Programms und der Laufzeit verglichen. Dabei werden sowohl die gesamte Laufzeit als auch die Laufzeit des Groundings betrachtet.

Die Ergebnisse zeigen, dass Body-Decoupled Grounding für normale Programme nicht so effizient wie für zyklenfreie Programme ist. In der Arbeit werden die Gründe dafür dargelegt und Verbesserungsvorschläge diskutiert. Andererseits geht aus dem Experiment hervor, dass die Variante mit codierter Ordnungsrelation schneller als die Variante mit geordneten Herleitungen ist.

Abstract

Answer set programming (ASP) is a programming methodology that is successfully applied in many real-world domains. One of the main reasons for that is the availability of efficient ASP systems. However, the grounding bottleneck still causes many difficulties, as the grounding process may increase the size of the input program substantially so that the ASP solver is not able to process the huge ground program.

Body-decoupled grounding is an approach that has been recently proposed to overcome this problem. It translates a non-ground normal program into a ground disjunctive program by decoupling predicates in the body of a rule and delegating their evaluation to the solver. The complexity of this translation is exponential only in the maximum predicate arity. In consequence, it is polynomial if the arity is bounded by a constant.

This work focuses on body-decoupled grounding in normal answer set programs. To this end, two variants of body-decoupled grounding for normal programs are presented. In comparison to the reduction for tight programs, these contain additional rules to ensure the foundedness of interpretations, particularly in case of programs with cycles. The practical feasibility of both variants are also evaluated experimentally. The goal of the experiment is to compare the implementations of both variants with each other, a state-of-the-art ASP system and another implementation, which is used in cycle-free programs, in terms of grounding size, grounding time and total runtime.

The experimental results demonstrate that body-decoupled grounding in normal programs is not as efficient as in tight programs. In the work, the reasons for that and possible improvements are discussed. On the other hand, the results also indicate that the variant with orderings outperforms the variant with ordered derivations.

Contents

Kurzfassung **v**

Abstract **vi**

Contents **vii**

1 Introduction **1**
1.1 Grounding Bottleneck 2
1.2 Contributions 3
1.3 Related Work 3

2 Preliminaries **4**
2.1 Ground ASP 4
2.2 Non-Ground ASP 8

3 Body-Decoupled Grounding via Solving **10**
3.1 Body-Decoupled Grounding for Tight ASP 10
3.2 Body-Decoupled Grounding for Normal ASP 14
3.3 Correctness 18
3.4 Complexity 19

4 Experiment **24**
4.1 Experimental Hypotheses 24
4.2 Setup 25
4.3 Benchmark Instances 25
4.4 Benchmark Measures 26
4.5 Results 26
4.6 Evaluation 27

5 Conclusion **30**

List of Figures **31**

List of Tables **32**

CHAPTER 1

Introduction

Answer set programming (ASP) is a declarative programming paradigm for modeling and solving search problems. In ASP, search problems are modeled in a specification language, and the models of the program, which are also called answer sets, correspond to the solutions of the problem [1].

The widely used formalism of the specification language is a type of logic programming language, and programs are evaluated by the stable model semantics [2]. In contrast to Prolog systems, ASP is fully declarative, i.e., changing the order of rules or literals does not effect the generated stable models [3]. On the one hand, this enables the ASP systems to optimize the programs easily and increases the efficiency, on the other hand, the development becomes more intuitive and faster [4].

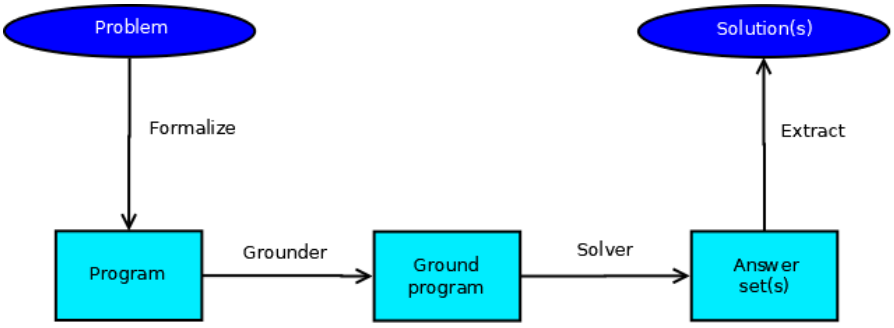

Figure 1.1: ASP paradigm with grounding and solving steps. Adapted from [1].

Most of the modern ASP systems work using two steps called grounding and solving. In order to represent a problem using the specification language of ASP, we first need to identify the problem and encode it as an ASP specification. Then, this specification is processed by a grounder so that it is translated into a ground program, i.e., variables are

replaced by constants. After that, a solver finds the answer sets in the ground program. In the last step, we interpret the answer sets to extract the solution. Figure 1.1 illustrates the process [1, 3, 4].

ASP originates from and is related to different areas like knowledge representation, non-monotonic reasoning, logic programming, constraint satisfaction and propositional satisfiability (SAT) [1, 3, 4]. The first ASP grounder *lparse* and solver *smodels* were implemented in 1996, after more than a decade since the start of research on stable model semantics [3]. Today, there are efficient systems with expressive specification languages available. As a consequence of this, ASP is successfully applied in a wide range of scientific and industrial domains such as natural language processing, phylogenetics, robotics, diagnosis, software engineering and many more [5, 6].

1.1 Grounding Bottleneck

Although there are efficient ASP systems available, the program may become substantially larger after grounding because each variable can usually be replaced by numerous possible constants. Therefore, the grounding step limits the efficiency of the system as the size of the ground program is too large for the solver that it cannot efficiently process the program and find the answer sets. This phenomenon is called *grounding bottleneck* in the literature [5].

A comparison between the complexity results of ground and non-ground programs explains the high costs of the grounding bottleneck for ASP systems. Compared to non-ground programs, the costs of ground programs are quite adequate as it is apparent from the complexity of consistency problem, which is the problem of deciding whether a program has an answer set [7]. The complexity is $\sum_2^P$-complete for the consistency of ground disjunctive programs [7] and may decrease to NP-complete for the consistency of ground normal programs [8]. However, it climbs up to $NEXPTIME$-completeness even for normal non-ground programs [9]. Fortunately, the complexity of grounding for non-ground programs can be limited under certain conditions. It has already been shown that the complexity of consistency for non-ground normal programs is $\sum_2^P$-complete if we assume that the predicate arity is fixed [10].

Utilizing this complexity result, a novel grounding approach, which decouples rule bodies during grounding and delegates their evaluation to the solver, has been proposed recently [11]. A reduction from non-ground tight programs to disjunctive programs is used to realize this approach. The reduction can also be extended to non-ground normal programs, where the notion of orderings (level mappings) is used in order to ensure justifiability. First experiments indicate that this approach is promising and can bring considerable speed-ups, especially for larger bodies with a dense structure.

1.2 Contributions

In this work, two variants of the reduction using body-decoupled grounding are presented that only differ in rules needed to ensure the justifiability for normal programs. Then, these variants are implemented in order to test experimentally if significant speed-ups for normal programs with cycles are also possible, which is already the case for cycle-free programs (tight programs) as demonstrated in [11]. Moreover, the experiment also enables to compare the performances of the implementation for tight programs and the implementations extended to normal programs by additional rules. Both variants are compared to each other and a modern ASP system in terms of grounding size, grounding time and total runtime.

The rest of this work is organized as follows. Chapter 1 is completed after giving an overview of the efficient grounding literature. Then, Chapter 2 continues with an introduction of the basics of ASP and some terms that are used in other chapters. In Chapter 3, the main ideas of body-decoupled grounding and corresponding reductions are presented. The experiments and results are reported in Chapter 4, where they are evaluated as well. The discussion, future work and conclusion follow in Chapter 5.

1.3 Related Work

The problem of grounding bottleneck is an active research area and has been studied by several works that have proposed various approaches to overcome it. Some techniques use traditional instantiation tactics and heuristics trying to optimize the algorithms used in standard ASP grounders [12, 13], while others try to estimate the grounding size of the program in order to support the decisions of ASP program rewriting systems so that optimized encodings based on the predictions are generated [14]. In addition, lazy grounding is an attempt to decrease the memory requirement by avoiding the grounding of the entire program before solving [15]. There are also alternative techniques to avoid grounding bottleneck including ASP modulo theory and constraint-programming [16, 17]. These techniques extend the traditional grounding by additional constructs or skipping the grounding step for some constraints. Finally, methods using bounded maximum predicate arity exist as well that propose novel grounding approaches [11, 18]. As previously mentioned, we will focus on the grounding procedure proposed by Besin et al. [11].

CHAPTER 2

Preliminaries

In this chapter, we introduce the basics of ASP, some terms, definitions and properties, which are useful for presenting the concepts in the next chapters. First, we define ground programs and answer sets. Then, using the definitons of ground ASP, we introduce tight programs and orderings (level mappings). After extending the definitions to non-ground ASP, we finish the chapter with the definition of grounding. Almost all terms and definitions are supported by examples in order to facilitate understanding. We adapted the majority of the information in this chapter from the paper [11].

2.1 Ground ASP

Let l, m and n be non-negative integers with $l \leq m \leq n$ and $a_1, \ldots, a_n$ be distinct propositional atoms. A *ground program* P is a set of rules of form

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

For each rule r:

- $H_r = \{a_1, \ldots, a_\ell\}$ denotes the set of head atoms occurring in r.
- $B_r^+ = \{a_{\ell+1}, \ldots, a_m\}$ denotes the set of positive body atoms occurring in r.
- $B_r^- = \{a_{m+1}, \ldots, a_n\}$ denotes the set of negative body atoms occurring in r.
- $at(r) = H_r \cup B_r^+ \cup B_r^-$ denotes the set of all atoms occurring in r.

Accordingly, $at(P) = \bigcup_{r \in P} at(r)$ denotes the set of all atoms occuring in rules of P.

A rule r is *normal* if $|H_r| \leq 1$, otherwise it is called *disjunctive*. A program P is called *normal* if all its rules are normal, otherwise *disjunctive*.

An *interpretation* I is a set of atoms that maps atoms to truth values, where a membership in I indicates that the atom has the truth value "true". I satisfies a rule r if at least one of the following two conditions is satisfied:

- $(H_r \cup B_r^-) \cap I \neq \emptyset$
- $B_r^+ \setminus I \neq \emptyset$

I is a *model* of P if it satisfies all rules of P.

In order to define an answer set, we use the *Gelfond-Lifschitz reduct* of P under I, which is the program P^I that can be obtained by doing the following in the given order [19]:

1. Remove all rules r with $B_r^- \cap I \neq \emptyset$, i.e., all rules with negative body atoms occurring in the interpretation are removed.
2. Remove all $\neg a$ where $a \in B_r^-$ from the remaining rules, i.e., all other negative literals are removed.

I is an *answer set* of a program P if I is a $\subseteq$*-minimal model* of P^I.

The *dependency graph* $\mathcal{D}_P$ of a program P is a directed graph containing the atoms of the set $\bigcup_{r \in P} H_r \cup B_r^+$ as vertices. For every vertex $a \in B_r^+$ and $b \in H_r$, there is an edge (a, b). P is *tight* if $\mathcal{D}_P$ does not have any directed cycle.

We can characterize the answer sets of normal programs as follows [20]:

Let I be a model of a normal program P and φ be an *ordering (level mapping)* function $\varphi : I \to \{0, \ldots, |I| - 1\}$ over I, which assigns a unique value to each atom in I from the set $\{0, \ldots, |I| - 1\}$. A rule $r \in P$ is *suitable for justifying* $a \in I$ if the following conditions are fulfilled:

- $a \in H_r$
- $B_r^+ \subseteq I$
- $B_r^- \cap I = \emptyset$
- $I \cap (H_r \setminus \{a\}) = \emptyset$

An atom $a \in I$ is *founded* if there is a rule r justifying a, which means:

- r is suitable for justifying a
- $\varphi(b) < \varphi(a)$ for every $b \in B_r^+$

If this is not the case, the atom a is unfounded. If every $a \in I$ is founded, I is also founded. In this regard, I is an answer set of a program P if

- I is a model of P.
- I is founded.

Ordering φ is not necessary for tight programs.

Example 1: *Consider the given program P_1 consisting of the rules r_1, r_2 and r_3:*

$$r_1 = b \leftarrow a, c$$
$$r_2 = c \leftarrow$$
$$r_3 = d \leftarrow \neg a$$

This program has the following properties:

- *For rule r_1*
  - $H_{r_1} = \{b\}$
  - $B^+_{r_1} = \{a, c\}$
  - $B^-_{r_1} = \emptyset$
  - $at(r_1) = \{a, b, c\}$
  - *r_1 is normal as $|H_{r_1}| = 1$*
  - *Interpretations like $I_{1_{r_1}} = \{c\}$ satisfy the rule r_1, while other interpretations like $I_{2_{r_1}} = \{d, a, c\}$ do not.*
- *For rule r_2*
  - $H_{r_2} = \{c\}$
  - $B^+_{r_2} = \emptyset$
  - $B^-_{r_2} = \emptyset$
  - $at(r_2) = \{c\}$
  - *r_2 is normal as $|H_{r_2}| = 1$*
  - *Interpretations like $I_{1_{r_2}} = \{c\}$ satisfy the rule r_2, while other interpretations like $I_{2_{r_2}} = \{b\}$ do not.*
- *For rule r_3*
  - $H_{r_3} = \{d\}$
  - $B_{r_3^+} = \emptyset$

  - $B_{r_3^-} = \{a\}$
  - $at(r_3) = \{a, d\}$
  - r_3 *is normal as* $|H_{r_3}| = 1$
  - *Interpretations like* $I_{1_{r_3}} = \{a, d\}$ *satisfy the rule* r_3*, while other interpretations like* $I_{2_{r_3}} = \{b, c\}$ *do not.*

- *For the entire program* P_1
  - $at(P_1) = \{a, b, c, d\}$
  - P_1 *is normal since all its rules are normal.*
  - $I_{1_{P_1}} = \{c, d, e\}$ *or* $I_{2_{P_1}} = \{c, d\}$ *are two of the models of* P_1*.*

In order to find the answer sets of P_1*, the Gelfond-Lifschitz reducts of* P_1 *under all interpretations are needed. An example under* $I_{2_{P_1}}$ *is given below.*
$P_1^{I_{2_{P_1}}}$ *consists of the rules* r_1*,* r_2 *and a modified version of* r_3 *because of the negated atom* a*, which is an element of* I*:*

$$r_1 = b \leftarrow a, c$$
$$r_2 = c \leftarrow$$
$$r_3' = d \leftarrow$$

As $I_{2_{P_1}}$ *is the only* $\subseteq$*-minimal model of* $P_1^{I_{2_{P_1}}}$*, it is the only answer set of* P_1*. We also point out that the foundedness of the atoms of* I*, namely* c *and* d*, can be derived easily by applying the definiton of foundedness above.*

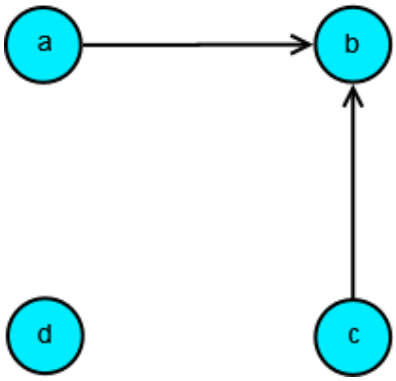

Figure 2.1: Cycle-free dependency graph $\mathcal{D}_{P_1}$.

The dependency graph $\mathcal{D}_{P_1}$ *is shown in Figure 2.1, which does not contain any cycles. Therefore,* P_1 *is a tight program. However, if we extend* P_1 *by adding another rule* $a \leftarrow b$*, we get a directed cycle between the vertices* a *and* b *as demonstrated in Figure 2.2 so that the extended program* P_1' *is not a tight program anymore.*

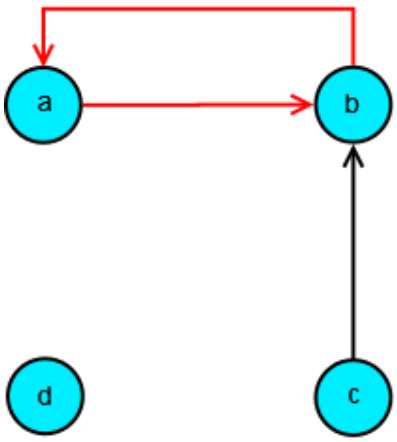

Figure 2.2: Dependency graph $\mathcal{D}_{P_1'}$ containing a directed cycle.

2.2 Non-Ground ASP

Before continuing with non-ground ASP, we clarify a notation detail: Let $X = \langle x_1, \ldots, x_m \rangle$ and $Y = \langle y_1, \ldots, y_n \rangle$ be vectors. We combine them by $\langle X, Y \rangle = \langle x_1, \ldots, x_m, y_1, \ldots, y_n \rangle$ and check if x_1 is an element of X by $x_1 \in X$.

A *(non-ground) program* Π is a set of *(non-ground) rules* of the form

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

where $p_1, \ldots, p_n$ are predicates and each of them has the arity $|p_i|$ for $1 \leq i \leq n$, which indicates the number of variables that a predicate takes. Thus, for every variable vector X_i, we have $|X_i| = |p_i|$. Safeness is also ensured, i.e., whenever $x \in \langle X_1, \ldots, X_\ell, X_{m+1}, \ldots, X_n \rangle$, then $x \in \langle X_{\ell+1}, \ldots, X_m \rangle$.

For each non-ground rule r:

- $H_r = \{p_1(X_1), \ldots, p_\ell(X_\ell)\}$ denotes the set of head predicates occurring in r.
- $B_r^+ = \{p_{\ell+1}(X_{\ell+1}), \ldots, p_m(X_m)\}$ denotes the set of positive body predicates occurring in r.
- $B_r^- = \{p_{m+1}(X_{m+1}), \ldots, p_n(X_n)\}$ denotes the set of negative body predicates occurring in r.
- $var(r) = \{x \in X \mid p(X) \in (H_r \cup B_r^+ \cup B_r^-)\}$ denotes the set of variables occurring in r.
- $||r|| = |H_r| + |B_r^+| + |B_r^-|$ denotes the rule size consisting of all predicates occurring in r.

For each program Π:

- For every two rules $r_1, r_2 \in \Pi$, it implies $var(r_1) \cap var(r_2) = \emptyset$ denoting that we assume that all variables are unique per rule (without loss of generality).
- $heads(\Pi) = \{p(X) \in H_r \mid r \in \Pi\}$ denotes all head predicates occurring in rules of Π.
- $hpreds(\Pi) = \{p \mid p(X) \in heads(\Pi)\}$ denotes the set of all predicate names occurring in rules of Π.
- $||\Pi|| = \sum_{r \in \Pi} ||r||$ denotes the program size consisting of all predicates occurring in rules of Π.

In order to ground a program Π, we need the following definitions:

- $\mathcal{F}$ denotes the set of facts, i.e., atoms of the form $p(D)$, where p is a predicate occurring in Π and D is a vector over domain values of size $|D| = |p|$.
- $dom(\Pi) = \{d \in D \mid p(D) \in \mathcal{F}\}$ denotes the domain of Π.
- $dom(X)$ is a domain vector over $dom(\Pi)$ for a variable vector X of size $|X|$.
- D_Y is a domain vector of D restricted to Y if D is a domain vector over variable vector X and vector Y contains only variables of X.

Utilizing the definitions above, we can get the grounded program $\mathcal{G}(\Pi)$ consisting of $\mathcal{F}$ and ground rules obtained by replacing each (non-ground) rule r for every domain vector $D \in dom(\langle var(r) \rangle)$ by

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

Finally, we define the variable graph $\mathcal{V}_\Pi$ for a non-ground program Π, whose vertices are the variables $var(r)$ of rule $r \in \Pi$, and there is an edge between two $x, y \in var(r)$ if a predicate $p(X)$ in r with $x, y \in X$ exists. Let X, Y be variable vectors. We define the reachable vertices in X by some $y \in Y$ in $\mathcal{V}_\Pi$ by $rch(Y, X)$.

Example 2: *Consider the given program* Π_1 *consisting of the rule* $r_4 = a(X, Y) \leftarrow b(X), c(Y, Z)$*, and the set of facts* $\mathcal{F} = \{b(1).\, c(1, 2).\}$*. First, we determine the domains of the variables:* $dom(X) = \{1\}$*,* $dom(Y) = dom(Z) = \{1, 2\}$*. The ground program* $\mathcal{G}(\Pi_1)$ *of* Π_1 *includes the following ground rules:*

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

The only answer set of the ground program $\mathcal{G}(\Pi_1)$ *is* $\{b(1), c(1,2), a(1,1)\}$.

CHAPTER 3

Body-Decoupled Grounding via Solving

In this chapter, the idea of body-decoupled grounding [11] is introduced. We present a reduction from a non-ground normal program into a ground disjunctive program. While the first part of the chapter covers only tight programs, the second section extends the reduction to normal programs and presents additional rules in order to ensure justifiability. We conclude the chapter with some properties and implications of the presented reduction procedure, in particular correctness and complexity.

3.1 Body-Decoupled Grounding for Tight ASP

First of all, we present the approach of body-decoupled grounding for cycle-free programs proposed by Besin et al. [11]. The reduction procedure $\mathcal{R}$ consists of three parts and translates a non-ground tight program Π into a ground disjunctive program $\mathcal{R}(\Pi)$ consisting of facts $\mathcal{F}$ and the following rules:

Guess Answer Set Candidates

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

$$h(D) \vee \bar{h}(D) \leftarrow \tag{3.1}$$

Ensure Satisfiability

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

$$\bigvee_{d \in dom(x)} sat_x(d) \leftarrow \tag{3.2}$$

for every $r \in \Pi, p(X) \in B_r^+, D \in dom(X), X = \langle x_1, \ldots, x_\ell \rangle$:
$$sat_r \leftarrow sat_{x_1}(D_{\langle x_1 \rangle}), \ldots, sat_{x_\ell}(D_{\langle x_\ell \rangle}), \neg p(D) \tag{3.3}$$

for every $r \in \Pi, p(X) \in H_r \cup B_r^-, D \in dom(X), X = \langle x_1, \ldots, x_\ell \rangle$:
$$sat_r \leftarrow sat_{x_1}(D_{\langle x_1 \rangle}), \ldots, sat_{x_\ell}(D_{\langle x_\ell \rangle}), p(D) \tag{3.4}$$

where $\Pi = \{r_1, \ldots, r_n\}$:
$$sat \leftarrow sat_{r_1}, \ldots, sat_{r_n} \tag{3.5}$$

for every $r \in \Pi, x \in var(r), d \in dom(x)$:
$$sat_x(d) \leftarrow sat \tag{3.6}$$

$$\leftarrow \neg sat \tag{3.7}$$

Prevent Unfoundedness

for every $r \in \Pi, h(X) \in H_r, D \in dom(X), y \in var(r), y \notin X$:
$$\bigvee_{d \in dom(y)} uf_y(\langle D, d \rangle) \leftarrow h(D) \tag{3.8}$$

for every $r \in \Pi, h(X) \in H_r, p(Y) \in B_r^+, D \in dom(\langle X,Y \rangle), Y = \langle y_1, \ldots, y_\ell \rangle$:
$$uf_r(D_X) \leftarrow uf_{y_1}(D_{\langle X, y_1 \rangle}), \ldots, uf_{y_\ell}(D_{\langle X, y_\ell \rangle}), \neg p(D_Y) \tag{3.9}$$

for every $r \in \Pi, h(X) \in H_r, p(Y) \in B_r^-, D \in dom(\langle X,Y \rangle), Y = \langle y_1, \ldots, y_\ell \rangle$:
$$uf_r(D_X) \leftarrow uf_{y_1}(D_{\langle X, y_1 \rangle}), \ldots, uf_{y_\ell}(D_{\langle X, y_\ell \rangle}), p(D_Y) \tag{3.10}$$

for every $h(X) \in heads(\Pi), D \in dom(X), \{r_1, \ldots, r_m\} = \{r \in \Pi \mid h(X) \in H_r\}$:
$$\leftarrow h(D), uf_{r_1}(D), \ldots, uf_{r_m}(D) \tag{3.11}$$

The first part of the procedure $\mathcal{R}$ has only one type of rule that is responsible for guessing the answer set candidates (Rules (3.1)). The second part ensures satisfiability while the third prevents unfoundedness.

In Rules (3.1), $\bar{h}(D)$ represents the negation of $h(D)$, for every $h(X) \in heads(\Pi)$ and $D \in dom(X)$, and chooses whether a head atom is in the answer set candidate or not. Only the Rules (3.2), which guess assignments of domain values to variables, cannot be modeled via choice rules; thus, disjunction is needed. Rules (3.3) and (3.4) ensure that a

rule $r \in \Pi$ is satisfiable if there is an assignment of domain values to variables in literals so that the resulting atom satisfies the rule. If we can derive the satisfiability of each rule $r \in \Pi$, we also derive the satisfiability of the entire program Π via Rule (3.5), whose derivation must be the case beacuse of the constraint in Rule (3.7). Moreover, Rules (3.6) assign all domain values to every variable, which is called *saturation*.

In order to prove the foundedness of a rule $r \in \Pi$, Rules (3.8) generate assignments of domain values to all variables for each head atom $h(D)$ contained in a model of $\mathcal{R}(\Pi)$, where $h(X) \in H_r$ and $D \in dom(X)$. Rules (3.9) and (3.10) are responsible for deriving the unfoundedness of a rule $r \in \Pi$, which is prevented by Rules (3.11).

We can also improve the rules for foundedness by making use of the variable independencies in order to get predicates with potentially smaller domain vectors for the rules (3.8), (3.9) and (3.10). Therefore, we use the variable graph as defined in Chapter 2 in order to utilize the variable independencies in the rules for justification. Given a non-ground program Π, an arbitrary rule $r \in \Pi$ with $h(X) \in H_r$ and $p(Y) \in B_r^+$, we let I be a set of atoms over the grounded program $\mathcal{G}(\Pi)$. We observe that if r does not justify $h(D_X) \in I$ for $D \in dom(\langle X, Y\rangle)$ due to $p(D_Y) \notin I$, r fails to justify any $h(D') \in I$ with $D' \in dom(X)$ and $D'_{\langle rch(Y,X)\rangle} = D_{\langle rch(Y,X)\rangle}$ as well. This leads to the new Rules (3.12), (3.13), (3.14) and (3.15) that replace the Rules (3.8), (3.9), (3.10) and (3.11):

Improved Foundedness, Replacing the Rules (3.8)-(3.11)

$$\text{for every } r \in \Pi, h(X) \in H_r, D \in dom(X), y \in var(r), y \notin X, p(Y) \in (B_r^+ \cup B_r^-), y \in Y: \quad \bigvee_{d \in dom(y)} uf_y(\langle D_{rch(Y,X)}, d\rangle) \leftarrow h(D) \tag{3.12}$$

$$\text{for every } r \in \Pi, h(X) \in H_r, p(Y) \in B_r^+, D \in dom(\langle X,Y\rangle), Y{=}\langle y_1,\ldots,y_\ell\rangle: \quad uf_{rch(Y,X)}(D_{\langle rch(Y,X)\rangle}) \leftarrow uf_{y_1}(D_{\langle rch(Y,X),y_1\rangle}),\ldots,uf_{y_\ell}(D_{\langle rch(Y,X),y_\ell\rangle}), \neg p(D_Y) \tag{3.13}$$

$$\text{for every } r \in \Pi, h(X) \in H_r, p(Y) \in B_r^-, D \in dom(\langle X,Y\rangle), Y{=}\langle y_1,\ldots,y_\ell\rangle: \quad uf_{rch(Y,X)}(D_{\langle rch(Y,X)\rangle}) \leftarrow uf_{y_1}(D_{\langle rch(Y,X),y_1\rangle}),\ldots,uf_{y_\ell}(D_{\langle rch(Y,X),y_\ell\rangle}), p(D_Y) \tag{3.14}$$

$$\text{for every } h(X) \in heads(\Pi), D \in dom(X), \{r_1,\ldots,r_m\} = \{r \in \Pi \mid h(X) \in H_r\}: \quad \leftarrow h(D), [\bigvee_{p(Y) \in B_{r_1}} uf_{rch(Y,X)}(D_{\langle rch(Y,X)\rangle})],\ldots,[\bigvee_{p(Y) \in B_{r_m}} uf_{rch(Y,X)}(D_{\langle rch(Y,X)\rangle})] \tag{3.15}$$

Rules (3.12), (3.13) and (3.14) have the only difference that the domain vectors of body and head rules (except the body of Rules (3.12)) are now potentially smaller, while Rules (3.15) have disjunctions in their bodies, which can be modeled via weight rules.

Example 3: *Consider the given program* Π_1 *consisting of the rule* $r_4 = a(X,Y) \leftarrow b(X), c(Y,Z)$*, and the set of facts* $\mathcal{F} = \{b(1).\ c(1,2).\}$*. First, we determine the domains*

of variables: $dom(X) = \{1\}$, $dom(Y) = dom(Z) = \{1, 2\}$. *The ground program* $\mathcal{R}(\Pi_1)$, *which is grounded using the reduction* $\mathcal{R}$ *explained in this section, includes the following rules:*

Rules (3.1) :
$a(1,1) \vee \bar{a}(1,1).\ a(1,2) \vee \bar{a}(1,2).$

Rules (3.2) :
$sat_X(1).\ sat_Y(1) \vee sat_Y(2).\ sat_Z(1) \vee sat_Z(2).$

Rules (3.3) :
$sat_{r_4} \leftarrow sat_X(1), \neg b(1).$
$sat_{r_4} \leftarrow sat_Y(1), sat_Z(1), \neg c(1,1).$
$sat_{r_4} \leftarrow sat_Y(1), sat_Z(2), \neg c(1,2).$
$sat_{r_4} \leftarrow sat_Y(2), sat_Z(1), \neg c(2,1).$
$sat_{r_4} \leftarrow sat_Y(2), sat_Z(2), \neg c(2,2).$

Rules (3.4) :
$sat_{r_4} \leftarrow sat_X(1), sat_Y(1), a(1,1).$
$sat_{r_4} \leftarrow sat_X(1), sat_Y(2), a(1,2).$

Rule (3.5) :
$sat \leftarrow sat_{r_4}.$

Rules (3.6) :
$sat_X(1) \leftarrow sat.\ sat_Y(1) \leftarrow sat.\ sat_Y(2) \leftarrow sat.$
$sat_Z(1) \leftarrow sat.\ sat_Z(2) \leftarrow sat.$

Rule (3.7) :
$\leftarrow \neg sat.$

Rules (3.8) :
$uf_Z(1,1,1) \vee uf_Z(1,1,2) \leftarrow a(1,1).$
$uf_Z(1,2,1) \vee uf_Z(1,2,2) \leftarrow a(1,2).$

Rules (3.9) :
$uf_{r_4}(1,1) \leftarrow \neg b(1).\ uf_{r_4}(1,1) \leftarrow \neg b(1).$

$uf_{r_4}(1,1) \leftarrow uf_Z(1,1,1), \neg c(1,1).\ uf_{r_4}(1,1) \leftarrow uf_Z(1,1,2), \neg c(1,2).$
$uf_{r_4}(1,2) \leftarrow uf_Z(1,2,1), \neg c(2,1).\ uf_{r_4}(1,2) \leftarrow uf_Z(1,2,2), \neg c(2,2).$

Rules (3.11) :
$\leftarrow a(1,1), uf_{r_4}(1,1).\ \leftarrow a(1,2), uf_{r_4}(1,2).$

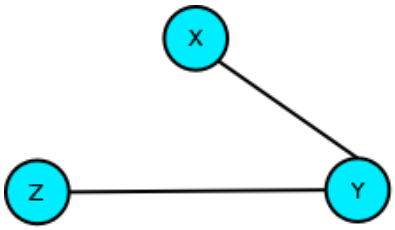

Figure 3.1: Variable graph $\mathcal{V}_{\Pi_1}$.

Using the variable graph $\mathcal{V}_{\Pi_1}$ *of* Π_1 *given in Figure 3.1, we get the following improved rules for foundedness replacing (3.8), (3.9) and (3.11):*

Rules (3.12) :
$uf_Z(1,1) \vee uf_Z(1,2) \leftarrow a(1,1).$
$uf_Z(2,1) \vee uf_Z(2,2) \leftarrow a(1,2).$

Rules (3.13) :
$uf_{r_{4_{\{X\}}}}(1) \leftarrow \neg b(1).\ uf_{r_{4_{\{X\}}}}(2) \leftarrow \neg b(2).$
$uf_{r_{4_{\{Y\}}}}(1) \leftarrow uf_Z(1,1), \neg c(1,1).\ uf_{r_{4_{\{Y\}}}}(2) \leftarrow uf_Z(2,1), \neg c(2,1).$
$uf_{r_{4_{\{Y\}}}}(1) \leftarrow uf_Z(1,2), \neg c(1,2).\ uf_{r_{4_{\{Y\}}}}(2) \leftarrow uf_Z(2,2), \neg c(2,2).$

Rules (3.15) :
$\leftarrow a(1,1), [uf_{r_{4_{\{X\}}}}(1) \vee uf_{r_{4_{\{Y\}}}}(1)].$
$\leftarrow a(1,2), [uf_{r_{4_{\{X\}}}}(1) \vee uf_{r_{4_{\{Y\}}}}(2)].$

The only answer set of the program $\mathcal{R}(\Pi_1)$ *restricted to symbols* a, b, c *of* Π_1 *is* $\{b(1), c(1,2), a(1,1)\}$ *as expected.*

3.2 Body-Decoupled Grounding for Normal ASP

So far, we have only considered the tight programs. However, the idea of body-decoupled grounding can be extended to normal programs. As normal programs may have directed cycles in their dependency graphs, we need to pay attention which atom is derived before

the other because the order of derivations plays a crucial role in guaranteeing foundedness in normal programs. The notion of level mappings that we introduced in Chapter 2 will help us to achieve this goal. In this section, we propose two ways of dealing with justifiability in programs containing cycles.

3.2.1 Variant 1: Using Orderings

The first variant is adapted from [11]. The idea is to encode derivation orderings by generating a quadratic number of auxiliary predicates in order to compare and store precedence. Three types of additional rules, which are listed below, model this idea:

Additional Rules for Foundedness of Normal Programs, Variant 1

$$\text{for every } p(X), p'(X') \in \mathit{heads}(\Pi), D \in \mathit{dom}(X), D' \in \mathit{dom}(X'), p(D) \neq p'(D'):$$
$$[p(D) \prec p'(D')] \vee [p'(D') \prec p(D)] \leftarrow \tag{3.16}$$

$$\text{for every } p_1(X_1), p_2(X_2), p_3(X_3) \in \mathit{heads}(\Pi), D_1 \in \mathit{dom}(X_1), D_2 \in \mathit{dom}(X_2), D_3 \in \mathit{dom}(X_3),$$
$$p_1(D_1) \neq p_2(D_2), p_2(D_2) \neq p_3(D_3), p_1(D_1) \neq p_3(D_3):$$
$$\leftarrow [p_1(D_1) \prec p_2(D_2)], [p_2(D_2) \prec p_3(D_3)], [p_3(D_3) \prec p_1(D_1)] \tag{3.17}$$

$$\text{for every } r \in \Pi, h(X) \in H_r, p(Y) \in B_r^+, D \in \mathit{dom}(\langle X,Y\rangle), Y = \langle y_1, \ldots, y_\ell\rangle, p(D_Y) \notin \mathcal{F}:$$
$$uf_r(D_X) \leftarrow uf_{y_1}(D_{\langle X,y_1\rangle}), \ldots, uf_{y_\ell}(D_{\langle X,y_\ell\rangle}), \neg[p(D_Y) \prec h(D_X)] \tag{3.18}$$

Rules (3.16) decide which head atom is derived before which other head atom in a program Π. For each atom $h(D)$, for which $h(X) \in \mathit{heads}(\Pi)$ and D is a domain vector over variable vector X, it is ensured that one of these atoms precedes the other. Rules (3.17) are responsible for transitivity, and they prevent three different atoms from preceding each other in a way that they form a cycle. Rules (3.18) are similar to Rules (3.9) and (3.10), which derive the unfoundedness of a rule. As the definition of foundedness of an atom a in an interpretation I does not allow that there exists a positive body atom $b \in B_r^+$ that is derived after a, where a is in the head of the rule r, it is impossible that a is founded if a precedes b, which is ensured by (3.18). Another point is that atoms in the set of facts $\mathcal{F}$ are the first ones derived. This is the reason why we do not need to compare them with other atoms.

Example 4: *Consider the given normal program* $\Pi_2 = \Pi_1 \cup \{r_5\}$ *with* $r_5 = c(Y,X) \leftarrow a(X,Y)$*, and the set of facts* $\mathcal{F} = \{b(1).\, c(1,2).\}$*. First, we determine the domains of the variables:* $\mathit{dom}(X) = \{1\}$*,* $\mathit{dom}(Y) = \mathit{dom}(Z) = \{1,2\}$*. The ground program* $\mathcal{R}(\Pi_2)$ *can be obtained by applying the reduction rules for tight programs as described in Section 3.1. However, additional rules to ensure justifiability in normal programs are also required. We give here only the additional rules using the Variant 1:*

Rules (3.16) :

$p_{a(1,1),c(1,1)} \vee p_{c(1,1),a(1,1)} \cdot p_{a(1,1),c(2,1)} \vee p_{c(2,1),a(1,1)} \cdot$

$p_{a(1,2),c(1,1)} \vee p_{c(1,1),a(1,2)} \cdot p_{a(1,2),c(2,1)} \vee p_{c(2,1),a(1,2)} \cdot$

Rules (3.18) :

$uf_{r_4}(1,1) \leftarrow uf_Z(1,1,1), \neg p_{c(1,1),a(1,1)} \cdot$

$uf_{r_4}(1,2) \leftarrow uf_Z(1,2,1), \neg p_{c(2,1),a(1,2)} \cdot$

$uf_{r_4}(1,2) \leftarrow uf_Z(1,2,2), \neg p_{c(2,2),a(1,2)} \cdot$

$uf_{r_5}(1,1) \leftarrow \neg p_{a(1,1),c(1,1)} \cdot$

$uf_{r_5}(2,1) \leftarrow \neg p_{a(1,2),c(2,1)} \cdot$

The only answer set of the program $\mathcal{R}(\Pi_2)$ *restricted to symbols* a, b, c *of* Π_2 *is* $\{b(1), c(1,2), a(1,1), c(1,1)\}$ *as expected.*

3.2.2 Variant 2: Proving via Ordered Derivations

The second variant does not use orderings directly. Instead, it proves by deriving each atom step by step, beginnig with facts, which are automatically proven, and continuing with other heads until a point is reached where no other derivations are possible. The acyclic nature of this idea prevents cycles between the new rules because it actually models a proving mechanism, where the order is preserved per definition. Contrary to the Variant 1, comparisons between the atoms of different rules are not required. The variant comprises five types of additional rules, three of them generated for each rule r in the original program Π and one of them for each $h(X) \in \mathit{heads}(\Pi)$. There is also another type of rule indicating that an atom a is automatically proven for each $a \in \mathcal{F}$. All extra rules are listed below:

Additional Rules for Foundedness of Normal Programs, Variant 2

for every $p(X) \in (H_r \cup B_r^+ \cup B_r^-), D \in \mathit{dom}(X), p(D) \in \mathcal{F}$:

$$prv_p(D) \leftarrow p(D) \tag{3.19}$$

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

$$jus_p(D_Y) \leftarrow uf_{y_1}(D_{\langle X,y_1 \rangle}), \ldots, uf_{y_\ell}(D_{\langle X,y_\ell \rangle}), prv_p(D_Y), p(D_Y). \tag{3.20}$$

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

$$jus_p(D_Y) \leftarrow uf_{y_1}(D_{\langle X,y_1 \rangle}), \ldots, uf_{y_\ell}(D_{\langle X,y_\ell \rangle}), \neg p(D_Y). \tag{3.21}$$

for every $r \in \Pi, h(X) \in H_r, (B_r^+ \cup B_r^-) = \{p_1, \ldots, p_n\}, D \in dom(\langle X, Y_1, \ldots, Y_n \rangle)$:
$$prv_h(D_X) \leftarrow jus_{p_1}(D_{Y_1}), \ldots, jus_{p_n}(D_{Y_n}). \tag{3.22}$$

for every $h(X) \in heads(\Pi), D \in dom(X)$:
$$\leftarrow h(D), \neg prv_h(D) \tag{3.23}$$

Since we have already clarified the reponsibility of the Rules (3.19), which indicate that facts are proven, we skip to the Rules (3.20) and (3.21), which are, respectively, responsible for deriving that positive body atoms are proven and negative body atoms are not. Furthermore, they ensure that a rule $r \in \Pi$ is suitable for justifying atom a if all other body atoms in the rule ensure that as well. The latter is derived by Rules (3.22), which prove the head using all proved bodies of the rule r that are able to justify the atom a. For this reason, the body of this atom is larger compared to other rules that come with the second variant. Rules (3.23) prevent unproven head atoms.

Example 5: *Consider the given normal program* Π_2 *described in more detail in example 4 again. This example has the only difference that we give the additional rules of Variant 2 instead of using the former one:*

Rules (3.19) :
$prv_b(1).$
$prv_c(1,2).$

Rules (3.20) :
$jus_{r1_{b(1)}} \leftarrow prv(1), b(1).$
$jus_{r1_{c(1,1)}} \leftarrow uf_Z(1,1), prv(1,1), c(1,1).$
$jus_{r1_{c(1,2)}} \leftarrow uf_Z(1,2), prv(1,2), c(1,2).$
$jus_{r1_{c(2,1)}} \leftarrow uf_Z(2,1), prv(2,1), c(2,1).$
$jus_{r1_{c(2,2)}} \leftarrow uf_Z(2,2), prv(2,2), c(2,2).$
$jus_{r2_{a(1,1)}} \leftarrow prv(1,1), a(1,1).$
$jus_{r2_{a(1,2)}} \leftarrow prv(1,2), a(1,2).$

Rules (3.22) :
$prv_a(1,1) \leftarrow jus_{r1_{b(1)}}, jus_{r1_{c(1,1)}}.$
$prv_a(1,1) \leftarrow jus_{r1_{b(1)}}, jus_{r1_{c(1,2)}}.$
$prv_a(1,2) \leftarrow jus_{r1_{b(1)}}, jus_{r1_{c(2,1)}}.$

$prv_a(1,2) \leftarrow jus_{r1_{b(1)}}, jus_{r1_{c(2,2)}}.$
$prv_c(1,1) \leftarrow jus_{r2_{a(1,1)}}.$
$prv_c(2,1) \leftarrow jus_{r2_{a(1,2)}}.$

Rules (3.23) :
$\leftarrow a(1,1), \neg prv_a(1,1).$
$\leftarrow a(1,2), \neg prv_a(1,2).$
$\leftarrow c(1,1), \neg prv_c(1,1).$
$\leftarrow c(2,1), \neg prv_c(2,1).$

The only answer set of the program $\mathcal{R}(\Pi_2)$ *restricted to symbols* a, b, c *of* Π_2 *is* $\{b(1), c(1,2), a(1,1), c(1,1)\}$ *as expected.*

3.3 Correctness

In this section, the correctness of body-decoupled grounding is shown. For this purpose, we give a proof idea. We also explain how the reduction can be applied only to some parts of the program without changing the results. This enables to use body-decoupled grounding only for rules that have large bodies, and the rest of the program can be grounded traditionally.

3.3.1 Correctness of Body-Decoupled Grounding for Tight Programs

Let Π be any non-ground tight program. Then, the grounding procedure $\mathcal{R}$ applied to Π is correct, i.e., the answer sets of $\mathcal{R}(\Pi)$ restricted to $at(\mathcal{G}(\Pi))$ correspond to the answer sets of $\mathcal{G}(\Pi)$. We can express that in a more precise way as well: For every answer set M' of $\mathcal{R}(\Pi)$, there is exactly one answer set $M' \cap at(\mathcal{G}(\Pi))$ of $\mathcal{G}(\Pi)$.

We give a proof idea, which is sketched in [11]:

We first prove that, for an answer set $M' \cap at(\mathcal{G}(\Pi))$ of $\mathcal{G}(\Pi)$, there is an answer set M' of $\mathcal{R}(\Pi)$. In order to prove that, we assume that M is an answer set of $\mathcal{G}(\Pi)$, but there is no $M' \supset M$ with $M' \cap at(\mathcal{G}(\Pi)) = M \cap at(\mathcal{G}(\Pi))$ that is an answer set of $\mathcal{R}(\Pi)$. This assumption leads to one of the following consequences:

- M is not a model of $\mathcal{G}(\Pi)$.
- M is unfounded.

The fulfillment of one of the two conditions above means a contradiction according to the definition of answer set in Chapter 2 because they imply that M cannot be an answer set of $\mathcal{G}(\Pi)$.

After proving the one side of the equivalence, we continue with the other side. To this end, we need to prove that $M' \cap at(\mathcal{G}(\Pi))$ is an answer set of $\mathcal{G}(\Pi)$ for an M' that is an answer set of $\mathcal{R}(\Pi)$. We assume that $M = M' \cap at(\mathcal{G}(\Pi))$ is not an answer set of $\mathcal{G}(\Pi)$. This leads to one of the following consequences:

- M' is not a model of $\mathcal{R}(\Pi)$.
- M' is unfounded.

If one of these conditions is satisfied, M' cannot be an answer set of $\mathcal{R}(\Pi)$. Therefore, we have a contradiction.

In this way, we confirm that the grounding procedure $\mathcal{R}$ is correct.

3.3.2 Partial Reducibility

The procedure $\mathcal{R}$ is not only correct, but it can also be applied to only a specific partition of a given program. For this reason, $\mathcal{R}$ can be applied in program parts where the most speed-up in comparison to standard grounding is achieved, e.g., rules with large bodies that cost a lot in terms of grounding size. The other parts are grounded traditionally so that the advantages of both grounding approaches are utilized efficiently. There are two propositions in [11] concerning partial reducibility. We summarize them in the following paragraphs.

The first proposition is about *partial reducibility of tight programs*: Let Π be any non-ground tight program. Given a partition of Π into programs Π_1 and Π_2 with $hpreds(\Pi_1) \cap hpreds(\Pi_2) = \emptyset$, the answer sets of $\mathcal{R}(\Pi_1) \cup \mathcal{G}(\Pi_2)$ restricted to $at(\mathcal{G}(\Pi))$ correspond to the answer sets of $\mathcal{G}(\Pi)$.

The second proposition deals with *partial reducibility of normal programs*: Let Π be any non-ground normal program. Given a partition of Π into programs Π_1 and Π_2 with $hpreds(\Pi_1) \cap hpreds(\Pi_2) = \emptyset$, where for every strongly connected component (SCC) C_1 of $\mathcal{D}_{\Pi_1}$ and SCC C_2 of $\mathcal{D}_{\Pi_2}$, C_1 and C_2 are disjoint, i.e., $C_1 \cap C_2 = \emptyset$. Then, the answer sets of $\mathcal{R}(\Pi_1) \cup \mathcal{G}(\Pi_2)$ restricted to $at(\mathcal{G}(\Pi))$ correspond to the answer sets of $\mathcal{G}(\Pi)$.

3.4 Complexity

This section is concerned with the complexity results of the body-decoupled grounding. First, we show the runtime and grounding size of the reduction $\mathcal{R}$ for tight programs. Second, we analyze the complexity of the additional rules for normal programs. In the following part, we analyze each rule separately. Our analysis is based on the number of rules generated by $\mathcal{R}$ and rule sizes. Note that Besin et al. [11] point out that disjunctive programs are inevitible because there cannot be a polynomial grounding procedure if a reduction $\mathcal{R}(\Pi)$ of program Π is normal.

3.4.1 Complexity of Body-Decoupled Grounding for Tight Programs

Let Π be a tight program and a the maximum predicate arity of Π.

Then, Rules (3.1) are constructed for each head $h(X) \in heads(\Pi)$ and $D \in dom(X)$. Moreover, there are at most $|dom(X)|^a$ many combinations for $h(D)$. Therefore, there are approximately $|heads(\Pi)| \cdot |dom(X)|^a$ many rules. As $|heads(\Pi)|$ is $\mathcal{O}(||\Pi||)$ and $|dom(X)|$ is $\mathcal{O}(|dom(\Pi)|)$, there are $\mathcal{O}(||\Pi|| \cdot |dom(\Pi)|^a)$ many rules. The rule size for (3.1) is always constant because there is always a disjunction of a head atom and its negation.

Rules (3.2) are constructed for every rule $r \in \Pi$ and $x \in var(r)$. As $|var(r)|$ is at most $||r|| \cdot a$ and Rules (3.2) are created for every rule r, there are at most $\mathcal{O}(||\Pi|| \cdot a)$ many rules. The rules comprise a disjunction of all $d \in dom(x)$, which means a size of $|dom(x)|$. Hence, the rule size for (3.2) is $\mathcal{O}(|dom(\Pi)|)$.

Rules (3.3) are constructed for every $r \in \Pi$, $p(X) \in B_r^+$, $D \in dom(X)$, where $X = \langle x_1, \ldots, x_\ell \rangle$. Moreover, there are at most $|dom(X)|^a$ many combinations for $p(D)$. Therefore, there are $|r| \cdot |B_r^+| \cdot |dom(X)|^a$ many rules. As $|r| \cdot |B_r^+|$ is $\mathcal{O}(||\Pi||)$ and $dom(X)$ is $\mathcal{O}(|dom(\Pi)|)$, there are $\mathcal{O}(||\Pi|| \cdot |dom(\Pi)|^a)$ many rules. The rule size is proportional to $|X|$, which is $\mathcal{O}(a)$; thus the rule size is $\mathcal{O}(a)$.

For (3.4), the same argumentation given in (3.3) applies. Therefore, the number of Rules (3.4) is $\mathcal{O}(||\Pi|| \cdot |dom(\Pi)|^a)$ and the size is $\mathcal{O}(a)$.

There is only one rule (3.5) consisting of a body containing all rules of Π. Therefore, the rule size is $\mathcal{O}(|\Pi|)$.

Rules (3.6) are constructed for every $r \in \Pi$, $x \in var(r)$, $d \in dom(x)$. This means that there are about $|\Pi| \cdot |var(r)| \cdot |dom(x)|$ many rules. As $|var(r)|$ is at most $||r|| \cdot a$ and Rules (3.6) are created for every rule r and member of $dom(x)$, there are at most $\mathcal{O}(||\Pi|| \cdot |dom(\Pi)|)$ many rules. The rules are of constant size because only two atoms are used per rule.

There is only one Rule (3.7) of constant size and it remains the same for every reduction.

Rules (3.8) are constructed for every $r \in \Pi$, $h(X) \in H_r$, $D \in dom(X)$ and $y \in var(r)$, where $y \notin X$. This leads to approximately $|\Pi| \cdot |H_r| \cdot |dom(X)| \cdot |var(r)|$ many rules. As $|r| \cdot |H_r|$ is $\mathcal{O}(||\Pi||)$, $dom(X)$ is $\mathcal{O}(|dom(\Pi)|)$ and $|var(r)|$ is at most $||r|| \cdot a$, there are $\mathcal{O}(||\Pi|| \cdot |dom(\Pi)|^a \cdot a)$ many rules. The rule size for (3.8) is $\mathcal{O}(|dom(\Pi)|)$ because there is always a disjunction of each $d \in dom(y)$.

Rules (3.9) are constructed for every $r \in \Pi$, $h(X) \in H_r$, $p(Y) \in B_r^+$ and $D \in dom(\langle X, Y \rangle)$, where $Y = \langle y_1, \ldots, y_\ell \rangle$. Moreover, there are at most $|dom(\langle X, Y \rangle)|^{2 \cdot a}$ many combinations for $uf_r(D_X)$ and $p(D_Y)$. Therefore, there are approximately $|\Pi| \cdot |H_r| \cdot |B_r^+| \cdot |dom(\langle X, Y \rangle)|^{2 \cdot a}$ many rules. As $|\Pi| \cdot |H_r| \cdot |B_r^+|$ is $\mathcal{O}(||\Pi||)$ and $|dom(\langle X, Y \rangle)|$ is $\mathcal{O}(|dom(\Pi)|)$, we get $\mathcal{O}(||\Pi|| \cdot |dom(\Pi)|^{2 \cdot a})$ many rules. The size of Rules (3.9) grows proportional to a, which implies that the rule size is $\mathcal{O}(a)$.

For (3.10), the same argumentation given in Rules (3.9) applies. Therefore, the number of Rules (3.10) is $\mathcal{O}(||\Pi|| \cdot dom(\langle X, Y\rangle)^{2 \cdot a})$ and the size is $\mathcal{O}(a)$.

Finally, Rules (3.11) are constructed for every $h(X) \in heads(\Pi)$, $D \in dom(X)$, where $\{r_1, \ldots, r_m\} = \{r \in \Pi \mid h(X) \in H_r\}$. Moreover, there are at most $|dom(X)|^a$ many combinations for $h(D)$. Therefore, there are approximately $|heads(\Pi)| \cdot |dom(X)|^a$ many rules. As $heads(\Pi)$ is $\mathcal{O}(||\Pi||)$ and $dom(X)$ is $\mathcal{O}(|dom(\Pi)|)$, there are $\mathcal{O}(||\Pi|| \cdot |dom(\Pi)|^a)$ many rules. The rule size for (3.11) is $\mathcal{O}(|\Pi|)$ as it is proportional to the number of rules in Π.

The number and size of Rules (3.12), (3.13), (3.14) and (3.15) match those of (3.8), (3.9), (3.10) and (3.11) respectively.

Table 3.1 summarizes the results for the rules (3.1)-(3.15). To sum up, we get the following result:

Proposition 1: The reduction procedure $\mathcal{R}$ applied on Π is polynomial if we assume that a is constant because it runs in time $\mathcal{O}(||\Pi|| \cdot |dom(\Pi)|^{2 \cdot a})$.

3.4.2 Complexity of Body-Decoupled Grounding for Normal Programs

We continue with the complexity results for additional rules of the variants for normal programs, which are described in detail in Section 3.2. Let Π be a normal program and a the maximum predicate arity of Π.

Then, Rules (3.16) are constructed for every $p(X)$, $p'(X') \in heads(\Pi)$, $D \in dom(X)$ and $D' \in dom(X')$, where $p(D) \neq p'(D')$. Moreover, there are at most $|dom(X)|^a$ many combinations for $p(D)$ and $|dom(X')|^a$ many combinations for $p'(D')$. Therefore, there are about $(|heads(\Pi)| \cdot |dom(X)|^a)^2$ many rules. As $|heads(\Pi)|$ is $\mathcal{O}(||\Pi||)$ and $|dom(X)|$, $|dom(X')|$ are $\mathcal{O}(|dom(\Pi)|)$, there are $\mathcal{O}((||\Pi|| \cdot |dom(\Pi)|^a)^2)$ many rules. The rule size is constant because there is always a disjunction of two auxiliary atoms that compare two head atoms.

Rules (3.17) are constructed for every $p_1(X_1), p_2(X_2), p_3(X_3) \in heads(\Pi)$, $D_1 \in dom(X_1)$, $D_2 \in dom(X_2)$ and $D_3 \in dom(X_3)$, where $p_1(D_1) \neq p_2(D_2)$, $p_2(D_2) \neq p_3(D_3)$ and $p_1(D_1) \neq p_3(D_3)$. Moreover, there are at most $|dom(X_1)|^a$ many combinations for $p_1(D_1)$, $|dom(X_2)|^a$ many combinations for $p_2(D_2)$ and $|dom(X_3)|^a$ many combinations for $p_3(D_3)$. Therefore, there are about $(|heads(\Pi)| \cdot |dom(X)|^a)^3$ many rules. As $|heads(\Pi)|$ is $\mathcal{O}(||\Pi||)$ and $|dom(X_1)|$, $|dom(X_2)|$, $|dom(X_3)|$ are $\mathcal{O}(|dom(\Pi)|)$, there are $\mathcal{O}((||\Pi|| \cdot |dom(\Pi)|^a)^3)$ many rules. The rule size is constant because there are three auxiliary atoms in the rule body.

Rules (3.18) are constructed for every $r \in \Pi$, $h(X) \in H_r$, $p(Y) \in B_r^+$ and $D \in dom(\langle X, Y\rangle)$, where $Y = \langle y_1, \ldots, y_\ell\rangle$ and $p(D_Y) \notin \mathcal{F}$. Moreover, there are at most $dom(\langle X, Y\rangle)^{2 \cdot a}$ many combinations for $uf_r(D_X)$ and $p(D_Y)$. Therefore, there are approximately $|\Pi| \cdot |H_r| \cdot |B_r^+| \cdot |dom(\langle X, Y\rangle)|^{2 \cdot a}$ many rules. As $|\Pi| \cdot |H_r| \cdot |B_r^+|$ is $\mathcal{O}(||\Pi||)$ and $|dom(\langle X, Y\rangle)|$ is $\mathcal{O}(|dom(\Pi)|)$, we get $\mathcal{O}(||\Pi|| \cdot |dom(\Pi)|^{2 \cdot a})$ many rules. The size of rules (3.18) grows proportional to a, which implies that the rule size is $\mathcal{O}(a)$.

Rules	Number	Size
(3.1)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|^a)$	$\mathcal{O}(1)$
(3.2)	$\mathcal{O}(\|\|\Pi\|\| \cdot a)$	$\mathcal{O}(\|dom(\Pi)\|)$
(3.3)	$\mathcal{O}((\|\|\Pi\|\| \cdot \|dom(\Pi)\|^a)$	$\mathcal{O}(a)$
(3.4)	$\mathcal{O}((\|\|\Pi\|\| \cdot \|dom(\Pi)\|^a)$	$\mathcal{O}(a)$
(3.5)	$\mathcal{O}(1)$	$\mathcal{O}(\|\Pi\|)$
(3.6)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|)$	$\mathcal{O}(1)$
(3.7)	$\mathcal{O}(1)$	$\mathcal{O}(1)$
(3.8)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|^a \cdot a)$	$\mathcal{O}(\|dom(\Pi)\|)$
(3.9)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|^{2 \cdot a})$	$\mathcal{O}(a)$
(3.10)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|^{2 \cdot a})$	$\mathcal{O}(a)$
(3.11)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|)$	$\mathcal{O}(\|\Pi\|)$
(3.12)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|^a \cdot a)$	$\mathcal{O}(\|dom(\Pi)\|)$
(3.13)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|^{2 \cdot a})$	$\mathcal{O}(a)$
(3.14)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|^{2 \cdot a})$	$\mathcal{O}(a)$
(3.15)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|)$	$\mathcal{O}(\|\Pi\|)$

Table 3.1: Complexity results for tight ASP.

Rules (3.19) are constructed for every $p(X) \in (H_r \cup B_r^+ \cup B_r^-)$, $D \in dom(X)$, where $p(D) \in \mathcal{F}$. Moreover, there are at most $|dom(X)|^a$ many combinations for $p(D)$. As $|\mathcal{F}|$ is $\mathcal{O}(||\Pi||)$ and $|dom(X)| \in \mathcal{O}(|dom(\Pi)|)$, there are $\mathcal{O}(||\Pi|| \cdot |dom(\Pi)|^a)$ many rules. The rule size is constant because it comprises only two atoms.

Rules (3.20) are constructed for every $r \in \Pi$, $h(X) \in H_r$, $p(Y) \in B_r^+$ and $D \in dom(\langle X, Y \rangle)$, where $Y = \langle y_1, \ldots, y_\ell \rangle$. Moreover, there are at most $dom(\langle X, Y \rangle)^a$ many combinations for $p(D_Y)$. Therefore, there are approximately $|\Pi| \cdot |H_r| \cdot |B_r^+| \cdot |dom(\langle X, Y \rangle)|^a$ many rules. As $|\Pi| \cdot |H_r| \cdot |B_r^+|$ is $\mathcal{O}(||\Pi||)$ and $|dom(\langle X, Y \rangle)|$ is $\mathcal{O}(|dom(\Pi)|)$, we get $\mathcal{O}(||\Pi|| \cdot |dom(\Pi)|^a)$ many rules. The size of rules (3.20) increases proportional to a, which implies that the rule size is $\mathcal{O}(a)$.

For (3.21), the same argumentation given in (3.20) applies. Therefore, the number of Rules (3.21) is $\mathcal{O}(||\Pi|| \cdot dom(\langle X, Y \rangle)^a)$ and the size is $\mathcal{O}(a)$.

Rules	Number	Size
(3.16)	$\mathcal{O}((\|\|\Pi\|\| \cdot \|dom(\Pi)\|^a)^2)$	$\mathcal{O}(1)$
(3.17)	$\mathcal{O}((\|\|\Pi\|\| \cdot \|dom(\Pi)\|^a)^3)$	$\mathcal{O}(1)$
(3.18)	$\mathcal{O}((\|\|\Pi\|\| \cdot \|dom(\Pi)\|^2 \cdot a)$	$\mathcal{O}(a)$
(3.19)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|^a)$	$\mathcal{O}(1)$
(3.20)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|^a)$	$\mathcal{O}(a)$
(3.21)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|^a)$	$\mathcal{O}(a)$
(3.22)	$\mathcal{O}(\|\Pi\| \cdot \|dom(\Pi)\|^{max(\|\|r_1\|\|,\ldots,\|\|r_n\|\|)\cdot a})$	$\mathcal{O}(max(\|\|r_1\|\|,\ldots,\|\|r_n\|\|))$
(3.23)	$\mathcal{O}(\|\|\Pi\|\| \cdot \|dom(\Pi)\|^a)$	$\mathcal{O}(1)$

Table 3.2: Complexity results for normal ASP.

Rules (3.22) are constructed for every $r \in \Pi$, $h(X) \in H_r$, $(B_r^+ \cup B_r^-) = \{p_1, \ldots, p_n\}$ and $D \in dom(\langle X, Y_1, \ldots, Y_n \rangle)$. For every $r \in \Pi$, we have at most $|dom(\langle X, Y \rangle)|^{||r||\cdot a}$ many combinations for $h(X), jus_{p_1}(D_{Y_1}), \ldots, jus_{p_n}(D_{Y_n})$. For the entire program Π, we get $\mathcal{O}(|\Pi| \cdot dom(\langle X, Y \rangle)^{max(||r_1||,\ldots,||r_n||)\cdot a})$. This implies that this rule makes Variant 2 exponential even under the assumption of a constant a because it also depends on the maximum rule size $max(||r_1||, \ldots, ||r_n||)$. The rule size is $\mathcal{O}(max(||r_1||, \ldots, ||r_n||))$ as all atoms in a rule r are used once.

Rules (3.23) are constructed for every $r \in \Pi$, $h(X) \in H_r$ and $D \in dom(X)$, where there are $|dom(X)|^a$ many combinations for $h(D)$. Therefore, there are approximately $|heads(\Pi)| \cdot |dom(X)|^a$ many rules. As $|heads(\Pi)|$ is $\mathcal{O}(||\Pi||)$ and $|dom(X)|$ is $\mathcal{O}(|dom(\Pi)|)$, there are $\mathcal{O}(||\Pi|| \cdot |dom(\Pi)|^a)$ many rules. The rule is constant as there are only two atoms are used.

Table 3.2 summarizes the results for the rules (3.16)-(3.23). To sum up, we get the following results:

Proposition 2: The reduction procedure $\mathcal{R}$ applied on Π is only for Variant 1 polynomial if we assume that a is constant because it runs in time $\mathcal{O}((||\Pi|| \cdot |dom(\Pi)|^a)^3)$ for the Variant 1 and $\mathcal{O}(|\Pi| \cdot dom(\langle X, Y \rangle)^{max(||r_1||,\ldots,||r_n||)\cdot a})$ for the Variant 2. In addition to the rules for tight programs, Variant 1, where we use orderings, contains the additional rules (3.16), (3.17) and (3.18), whereas Variant 2 contains (3.19), (3.20), (3.21), (3.22) and (3.23).

CHAPTER 4

Experiment

In this chapter, we present the experimental results to evaluate the feasibility of the body-decoupled grounding approach. For this purpose, we refactored and extended the implementation of the tool *newground* [11] so that it can apply the reduction to normal programs with cycles as well. We implemented two different variants for ensuring justifiability in normal programs as we have already described in detail in Chapter 3. Both variants of the tool are written in *Python3* and use the API of *clingo 5.5*, particularly the parsing functionality of logic programs using syntax trees. They also enable partial reducibility in order to allow users to specify which parts of the program should be grounded via body-decoupled grounding and whether the additional rules for normal programs should be used. Precompiling of usual compare operators is also supported with the aim of getting more compact programs.

The tools, instances and supplemental material of this work are publicly available at *https://github.com/k4u6/newground2.*

4.1 Experimental Hypotheses

In the experiment, we study the following hypotheses:

1. In contrast to traditional grounding, our new tool *newground2* suffers less from large instances that may run into grounding bottlenecks, as it was the case for *newground* in [11].

2. Body-decoupled grounding for tight programs can massively reduce grounding sizes and grounding times [11]. Body-decoupled grounding for normal programs can preserve those grounding sizes and grounding times despite the additional rules.

3. Body-decoupled grounding for tight programs can improve overall performance (grounding and solving) [11]. Body-decoupled grounding for normal programs can preserve the performance despite the additional rules.

4. Body-decoupled grounding with additional rules for normal programs efficiently operates with other approaches like body-decoupled grounding for tight programs does [11].

5. As the theoretical runtime and grounding size of Variant 2 are greater than Variant 1, Variant 1 outperforms Variant 2 in terms of grounding size, grounding time and overall time.

4.2 Setup

We implemented the tool *newground2* that extends *newground* by additional rules for normal programs. As we implemented two different variants, we refer to the Variant 1, where we use orderings, as *newground2-v1* and to the Variant 2, where we use ordered derivations for provability, as *newground2-v2* in the rest of the chapter. All experiments were conducted on a computer with Intel Core i7-8565U processor at 1.8 GHz clock speed and 16 GB RAM using *Python 3.10.5* and *clingo 5.5.2*. We compare the performances of the following tools:

- *gringo* version 5.5.2.
- *newground2-tight*: without additional rules for normal programs.
- *newground2-v1*: Variant 1.
- *newground2-v2*: Variant 2.

For *newground2-tight*, body-decoupled grounding is applied only on certain non-ground rules that potentially cause grounding overhead. However, for *newground2-v1* and *newground2-v2*, some other rules must be added because of the additional conditions for partial reducibility in normal programs (see Section 3.3).

4.3 Benchmark Instances

In order to answer the hypotheses, we study the non-partitional-removal-colorings (nprc) problem, whose encoding is taken from [15]. This problem checks whether there is a possible assignment of (three) colors to each vertex with an exception of one vertex, which is not colored and does not partition the graph so that each pair of adjacent vertices is colored with a different color. All instances consist of randomly generated instances from 5 to 1200 vertices, with an edge probability (density) from 0.1 to 1.0.

4.4 Benchmark Measures

We measure the grounding sizes, grounding times and overall times. Overall time means the sum of grounding time and solving time. For computing grounding sizes, we compute the size of the output generated from the respective tool. In order to achieve this for *gringo*, we use it with the option *--verbose=2 --output text*. For overall performance, we use *clingo* with *-q --stats=2*, i.e., one answer set is computed. In addition, we append *--project* for all variants of *newground2* to ensure answer sets are over the same set of atoms and there are no redundant solutions because of the additional atoms generated by *newground2*.

4.5 Results

In this section, we demonstrate the results of the experiment. This section is divided into three parts, each dedicated to one of the benchmark measures of the experiment: grounding scalability, grounding performance and overall performance.

4.5.1 Grounding Scalability

Figure 4.1 depicts the grounding sizes. However, the sizes are not given if grounding takes more than 1200 seconds. It is obvious that *newground-tight* dominates the other tools with a big difference so that we only show results up to the instance size 600 in order to present the results for the other tools clearly. While *gringo* takes the second place despite problems with large instances, *newground-v1* and *newground-v2* can only deal with the smallest instances. The problem of *newground-v1* is that grounding size explodes with the increasing instance size, whereas *newground-v2*, which is not even easily visible on the plot, has the problem that grounding takes too much time that grounding sizes are only available for the smallest instance. If Rules (3.22) were implemented with a more suitable data structure and without redundant computations, it would perform better but could not beat even one of the tools because Rules (3.22) consume too much time and are inherently complex. All these results falsify the hypothesis 1.

4.5.2 Grounding Performance

Figure 4.2 illustrates the results for grounding performance. While *newground-tight* clearly outperforms the other tools and grounds every instance, other tools cannot finish some instances in 1200 seconds, like *gringo*, which has the second best grounding performance. The situation is worse for *newground2-v1* and *newground-v2*, but *newground2-v1* grounds obviously faster than *newground2-v2*. As the results are proportional to those of grounding size, they are plausible and falsify the hypotheses 1 and 2.

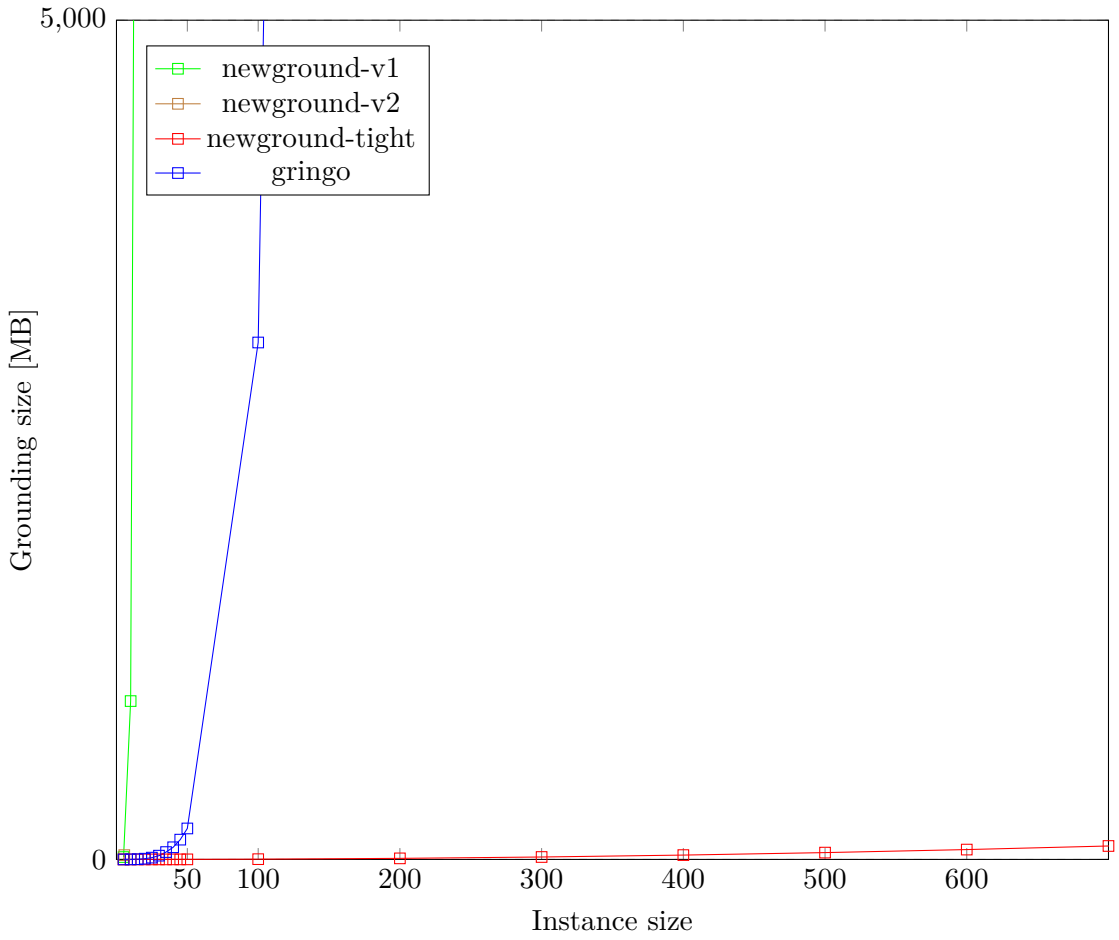

Figure 4.1: Grounding size for *newground-v1*, *newground-v2*, *newground-tight* and *gringo*. Instance size is the number of graph vertices with a density of 0.5.

4.5.3 Overall Performance

Regarding the overall performance, Figure 4.3 demonstrates that *newground-tight* is much more efficient than the other tools. However, *newground-v1* and *newground-v2* cannot preserve the performance of *newground-tight* again. Therefore, *gringo* is faster than both versions. Overall, we cannot confirm the hypothesis 3 as well.

4.6 Evaluation

Results show that *newground-tight* is more efficient than standard ASP systems. However, the efficiency decreases and is much worse than *gringo* if additional rules for normal programs are used. One reason for that is the non-optimal implementation of the additional rules. For example, we could avoid some redundant rules for Rules (3.16)

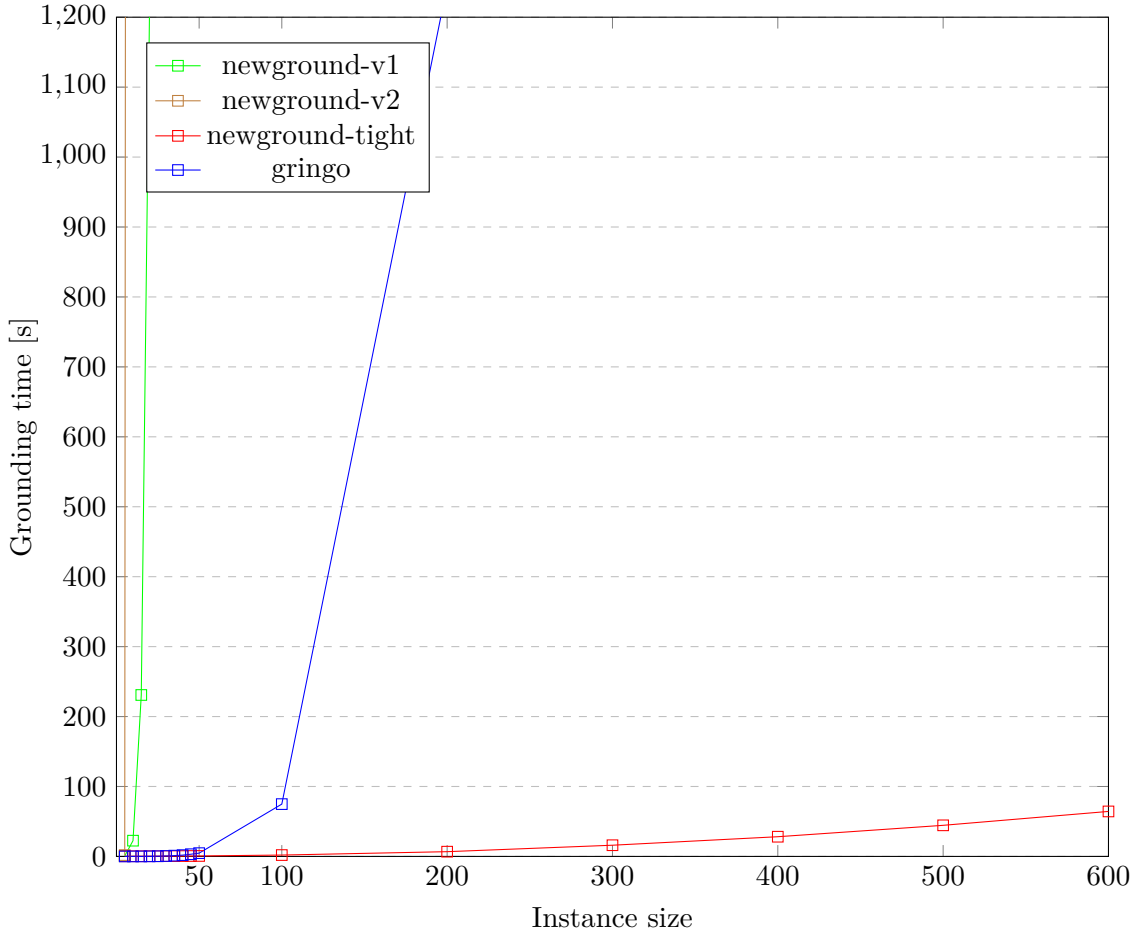

Figure 4.2: Grounding performance for *newground-v1*, *newground-v2*, *newground-tight* and *gringo*. Instance size is the number of graph vertices.

and Rules (3.22) could be implemented with a more suitable data structure or could be grounded by *clingo*.

In fact, the results are not unexpected if we consider the complexity results. Moreover, we are more flexible in tight programs while we are choosing the program parts to decouple their rule bodies. For normal programs, we need to consider that there are no strongly connected components (SCC) that overlap. All these factors lead to the consequence that these rules need to be optimized in order to utilize the body-decoupled grounding for normal programs efficiently.

One possible improvement could be to determine the SCCs of dependency graphs before the reduction so that the non-overlapping SCCs can be grounded almost independently. This may help in dropping the rule number and size because additional rules would not be generated for predicates or rules that cannot build a cycle with each other.

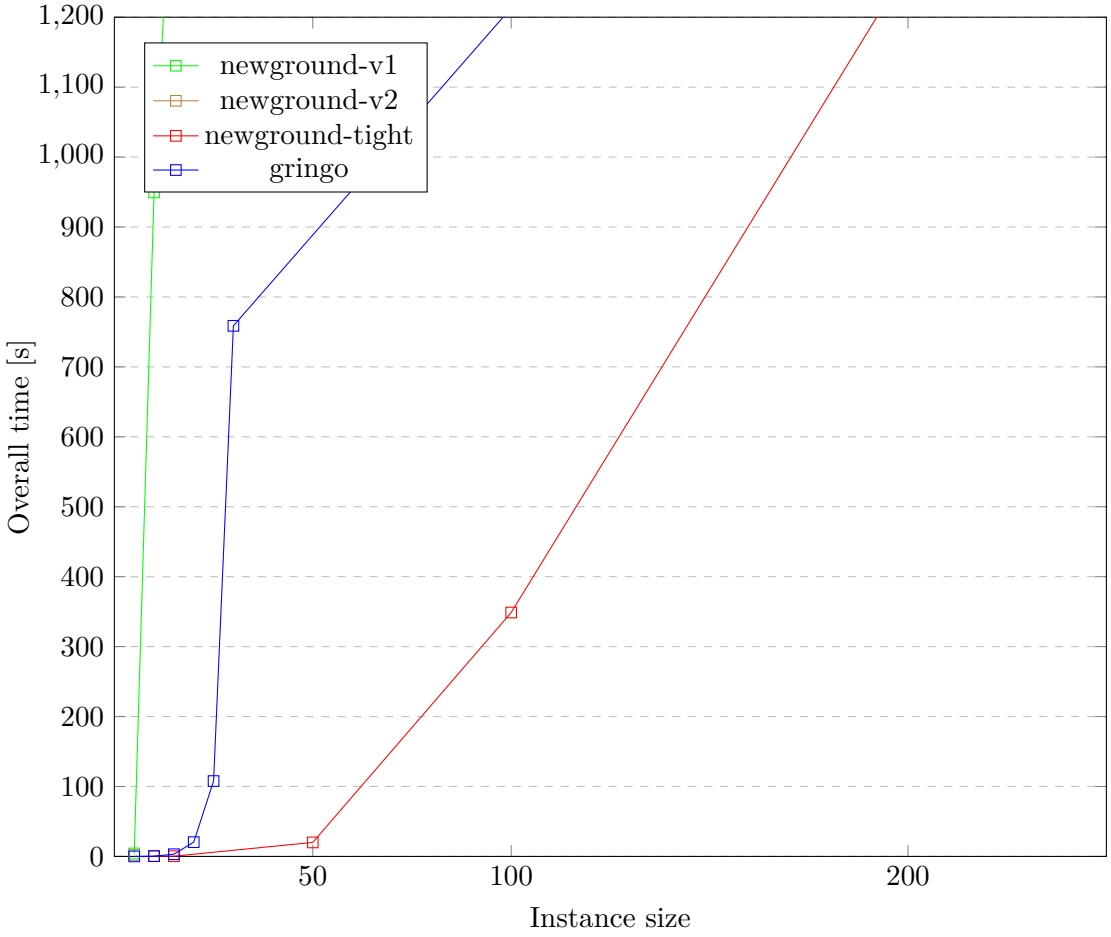

Figure 4.3: Overall performance for *newground-v1*, *newground-v2*, *newground-tight* and *gringo*. Instance size is the number of graph vertices.

Finally, we can only confirm the hypothesis 5, as Variant 1 performed better than Variant 2. We can also say that the hypothesis 4 is also confirmed partially because body-decoupled grounding for normal programs operates with other approaches but not in an efficent way.

CHAPTER 5

Conclusion

In this work, we analyzed the body-decoupled grounding via search, a novel approach to overcome the grounding bottleneck. First, we presented the approach and some of its properties. Although the main focus lies on normal programs, we also analyzed tight programs. Then, we evaluated the practical performance of body decoupled grounding experimentally. The results demonstrate that body decoupled grounding brings considerable speed-ups if it is applied to tight program parts, especially to those parts containing rules with large bodies and many variables. However, in accordance with its higher computational complexity and limited partial reducibility compared to tight programs, body decoupled grounding applied to normal program parts is not as efficient as it is for tight programs. In particular, the second variant, where rules are proved via ordered derivations, runs in exponential time even if the maximum predicate arity is fixed; thus it has a negative impact on performance. The first variant, where we use auxiliary predicates to encode and compare orderings, has polynomial complexity under the assumption of constant maximum predicate arity.

Nevertheless, we plan to optimize the body-decoupled grounding procedure by encoding orderings in a more compact way. For instance, strongly connected components may be determined before the reduction and they can be grounded almost independently. This may help in dropping the rule number and size. Besides, there are also other improvements of the reduction procedure to consider.

As the approach works at least for tight programs well and outperforms a modern grounder (gringo), it would be interesting to integrate it into intelligent grounders [21] so that they recognize when to use body decoupled grounding. In suitable situations, e.g. a program with multiple non-overlapping strongly connected components, an optimized version for normal programs may be useful and combined with the variant for tight programs.

List of Figures

1.1 ASP paradigm with grounding and solving steps. Adapted from [1]. . . . 1

2.1 Cycle-free dependency graph $\mathcal{D}_{P_1}$. 7
2.2 Dependency graph $\mathcal{D}_{P_1'}$ containing a directed cycle. 8

3.1 Variable graph $\mathcal{V}_{\Pi_1}$. 14

4.1 Grounding size for *newground-v1*, *newground-v2*, *newground-tight* and *gringo*. Instance size is the number of graph vertices with a density of 0.5. 27
4.2 Grounding performance for *newground-v1*, *newground-v2*, *newground-tight* and *gringo*. Instance size is the number of graph vertices. 28
4.3 Overall performance for *newground-v1*, *newground-v2*, *newground-tight* and *gringo*. Instance size is the number of graph vertices. 29

List of Tables

3.1 Complexity results for tight ASP. 22
3.2 Complexity results for normal ASP. 23

Bibliography

[1] T. Janhunen and I. Niemelä, "The answer set programming paradigm," *AI Magazine*, vol. 37, no. 3, pp. 13–24, 2016.

[2] F. Calimeri, W. Faber, M. Gebser, G. Ianni, R. Kaminski, T. Krennwallner, N. Leone, M. Marco, F. Ricca, and T. Schaub, "Asp-core-2 input language format," *Theory and Practice of Logic Programming*, vol. 20, no. 2, pp. 294–309, 2020.

[3] V. W. Marek, I. Niemelä, and M. Truszczynski, "Origins of answer-set programming - some background and two personal accounts," *CoRR*, vol. abs/1108.3281, 2011.

[4] G. Brewka, T. Eiter, and M. Truszczynski, "Answer set programming at a glance," *Communications of the ACM*, vol. 54, no. 12, pp. 92–103, 2011.

[5] A. Falkner, G. Friedrich, K. Schekotihin, R. Taupe, and E. C. Teppan, "Industrial applications of answer set programming," *Künstliche Intelligenz*, vol. 32, no. 2-3, pp. 165–176, 2018.

[6] E. Erdem, M. Gelfond, and N. Leone, "Applications of answer set programming," *AI Magazine*, vol. 37, no. 3, pp. 53–68, 2016.

[7] T. Eiter and G. Gottlob, "On the computational cost of disjunctive logic programming: Propositional case," *Annals of Mathematics and Artificial Intelligence*, vol. 15, no. 3-4, pp. 289–323, 1995.

[8] V. W. Marek and M. Truszczynski, "Autoepistemic logic," *Journal of the ACM*, vol. 38, no. 3, pp. 588–619, 1991.

[9] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov, "Complexity and expressive power of logic programming," *ACM Comput. Surv.*, vol. 33, no. 3, pp. 374–425, 2001.

[10] T. Eiter, W. Faber, M. Fink, and S. Woltran, "Complexity results for answer set programming with bounded predicate arities and implications," *Annals of Mathematics and Artificial Intelligence*, vol. 51, no. 2-4, pp. 123–165, 2007.

[11] V. Besin, M. Hecher, and S. Woltran, "Body-decoupled grounding via solving: A novel approach on the asp bottleneck," in *Proceedings of the 31st International Joint Conference on Artificial Intelligence, IJCAI 2022*, 2022, in Press.

[12] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Multi-shot asp solving with clingo," *Theory and Practice of Logic Programming*, vol. 19, no. 1, pp. 27–82, 2019.

[13] R. Kaminski and T. Schaub, "On the foundations of grounding in answer set programming," *CoRR*, vol. abs/2108.04769, 2021.

[14] N. Hippen and Y. Lierler, "Estimating grounding sizes of logic programs under answer set semantics," in *Logics in Artificial Intelligence - 17th European Conference, JELIA 2021, Virtual Event, May 17-20, 2021, Proceedings*, vol. 12678. Springer, 2021, pp. 346–361.

[15] A. Weinzierl, R. Taupe, and G. Friedrich, "Advancing lazy-grounding asp solving techniques - restarts, phase saving, heuristics, and more," *Theory and Practice of Logic Programming*, vol. 20, no. 5, pp. 609–624, 2020.

[16] M. Banbara, B. Kaufmann, M. Ostrowski, and T. Schaub, "Clingcon: The next generation," *Theory and Practice of Logic Programming*, vol. 17, no. 4, pp. 408–461, 2017.

[17] P. Cabalar, J. Fandinno, T. Schaub, and P. Wanko, "A uniform treatment of aggregates and constraints in hybrid asp," in *Proceedings of the 17th International Conference on Principles of Knowledge Representation and Reasoning, KR 2020, Rhodes, Greece, September 12-18, 2020*, 2020, pp. 193–202.

[18] T. Eiter, W. Faber, and M. Mushthofa, "Space efficient evaluation of ASP programs with bounded predicate arities," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11-15, 2010*. AAAI Press, 2010.

[19] M. Gelfond and V. Lifschitz, "Classical negation in logic programs and disjunctive databases," *New Generation Computing*, vol. 9, no. 3/4, pp. 365–386, 1991.

[20] F. Lin and J. Zhao, "On tight logic programs and yet another translation from normal logic programs to propositional logic," in *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, August 9-15, 2003*. Morgan Kaufmann, 2003, pp. 853–858.

[21] F. Calimeri, D. Fuscà, S. Perri, and J. Zangari, "I-DLV: the new intelligent grounder of DLV," *Intelligenza Artificiale*, vol. 11, no. 1, pp. 5–20, 2017.