MAGISTERARBEIT

Techniques for Simplifying Disjunctive Datalog Programs with Negation

ausgeführt am Institut für Informationssysteme
Abteilung Wissensbasierte Systeme 184/3
der Technischen Universität Wien

unter der Anleitung von

O. Univ. Prof. Dipl.-Ing. Dr. techn. Thomas Eiter

und

Dipl.-Ing. Dr. techn. Stefan Woltran
als verantwortlich mitwirkendem Universitätsassistenten

durch

Patrick Traxler
Matrikelnr. 0027287
Rudmanns 132, 3910 Zwettl

Wien, am 12. Jänner 2006 .....

Deutsche Zusammenfassung

In dieser Diplomarbeit führen wir Techniken zur Vereinfachung disjunktiver Datalogprogramme mit Negation ein und studieren deren Komplexität. Disjunktive Datalogprogramme mit Negation werden dazu verwendet, Probleme unterschiedlicher Art zu beschreiben. Dazu zählen etwa kombinatorische Probleme, Planungsprobleme sowie Probleme aus der Künstlichen Intelligenz. Ein Datalogprogramm bestimmt dabei die Lösungen einer Probleminstanz, ohne auf deren Berechnung einzugehen. Daher spricht man von *deklarativer Programmierung*.

In manchen Situationen ist es sinnvoll Datalogprogramme zu vereinfachen, zum Beispiel wenn dadurch die Auswertung des Programms beschleunigt wird. Vereinfachung ist insbesondere für automatisch generierte Programme interessant, da diese redundante Programmteile enthalten können. Zum Beispiel generiert das System DLV$^{\mathrm{K}}$ aus der Beschreibung eines Plannungsproblems automatisch ein Datalogprogramm, das seinerseits durch das System DLV[1] ausgewertet werden kann. Auch für das theoretische Arbeiten mit Datalogprogrammen bringt es Erleichterungen mit sich, vereinfachte Programme zu betrachten.

Zwei Formen der Vereinfachung werden vorgestellt: Regelelimination und Regeltransformation. Ein Datalogprogramm ist eine Menge von Regeln. Eine Regel kann entfernt werden, wenn dabei Äquivalenz erhalten bleibt. Wir betrachten hierfür *gewöhnliche Äquivalenz*, *starke Äquivalenz* und *uniforme Äquivalenz*. Diese und andere notwendige Begriffe werden im Kapitel 2 präsentiert. Kapitel 3 widmet sich der Regelelimination und Regeltransformation wird im Kapitel 4 behandelt. Regeltransformation bezeichnet die Umformung einer Regel in eine oder mehrere strukturell einfachere Regeln, sodass das resultierende Programm äquivalent zum ursprünglichen

[1] http://www.dlvsystem.com

Programm ist. Kapitel 5 behandelt Aspekte der Implementierung. Offene Fragen und weiterführende Arbeiten werden im Kapitel 6 besprochen. Eine Zusammenfassung der Resultate findet sich im Kapitel 7.

Die erzielten Resultate umfassen eine Menge von Techniken. Wir beweisen deren Korrketheit, d.h., die angewandten Techniken erhalten Äquivalenz. Außerdem analysieren wir die Komplexität der Techniken, zeigen also, mit welchem rechnerischem Aufwand Regelelimination und Regeltransformation durchgeführt werden können. Einige der Probleme sind in polynomieller Zeit lösbar, andere dagegen vollständig für NP bzw. für PSPACE. Die komplexitätstheoretischen Grundbegriffe werden ebenfalls in Kapitel 2 eingeführt. Die wichtigsten Resultate wurden in [E+06] und [EFT05] veröffentlicht.

Abstract

In this thesis we introduce techniques for simplifying disjunctive datalog programs with negation and study their complexity. Disjunctive datalog programs with negation are used to describe problems of different kind. For example, combinatorial problems, planning problems, and problems of Artificial Intelligence. A datalog program determines the solutions of a problem instance without saying anything about the way of computing it. Therefore, we speak of *declarative programming*.

In some situations it is useful to simplify datalog programs, e.g., if the evaluation of the program becomes faster. Simplification is especially interesting for automatically generated programs since these programs may contain large redundant parts. For example, the system DLV^{K} generates from a description of a planning problem automatically a datalog program which then can be evaluated by the system DLV[2]. But also for theoretical considerations it is often preferable to use simplified programs.

Two kinds of simplification are presented: rule elimination and rule transformation. A datalog program is a set of rules. A rule may be eliminated if equivalence is preserved. We are considering *ordinary equivalence*, *strong equivalence*, and *uniform equivalence*. The necessary notions will introduced in Chapter 2. Chapter 3 concerns rule elimination and rule transformation will be studied in Chapter 4. Rule transformation is the rewriting of a rule into one or more structural simpler rules such that the resultant program is equivalent to the original program. Chapter 5 concerns aspects of implementations. Open questions and further work are discussed in Chapter 6. A summarization of the results can be found in Chapter 7.

Our results encompass several techniques. We are proving their correct-

[2]http://www.dlvsystem.com

ness, i.e., the techniques preserve equivalence. Moreover, we analyze their complexity, that is, we analyze the computational resources needed for rule elimination and rule transformation. Some of the problems are solvable in polynomial time, others are complete for NP or PSPACE. The complexity theoretic notions are introduced in Chapter 2. Our most important results have been published in [E+06] and [EFT05].

Contents

Chapter 1

Introduction

In this thesis disjunctive datalog programs with negation are considered. The semantics of disjunctive datalog programs with negation is based upon the stable model semantics of propositional datalog programs with negation as failure which was introduced by Gelfond and Lifschitz [GL91]. A disjunctive datalog program with negation consists of rules of the form

$$a_1 \vee ... \vee a_k \leftarrow a_{k+1}, ..., a_m, \text{not } a_{m+1}, ..., \text{not } a_n$$

where the a_i's are atoms with variables. Intuitively, whenever the body of the rule (right side) is satisfied then the head (left side) has to be satisfied too.

Simplifications of disjunctive datalog programs with negation are useful in many situations. Simplifications can be considered to

- make the program smaller,
- simplify the structure of the program, or
- speed up the evaluation of the program.

Because of this latter important case we also speak of optimization. The major aim of this thesis is to study program simplification and optimization, and to analyze the complexity of the used techniques in particular.

Simplification involves that the resultant program is equivalent to the original program. There are several notions of equivalence. Two important ones in the area of logic programs are strong and uniform equivalence. The

notion of strong equivalence in the propositional case was introduced by Lifschitz, Pearce and Valverde [LPV01], and the notion of uniform equivalence by Eiter and Fink [EF03] which based upon [Sagiv88, Maher88]. Similar notions for the non-ground case – the programs may have variables – were studied by Eiter et al. [EFTW05]. These notions are used instead of ordinary equivalence because ordinary equivalence is a rather weak notion for disjunctive datalog programs with negation.

The considered techniques fall into two categories.

- *Rule elimination*: Under which circumstances can a rule r be eliminated from a program Π such that Π is strongly equivalent/uniformly equivalent to $\Pi - \{r\}$.
- *Rule transformation*: Under which circumstances can a rule r be replaced in a program Π by rules $r'_1, ..., r'_n$ stemming from a syntactically more restricted setting than r such that Π is strongly equivalent/uniformly equivalent to $\Pi - \{r\} \cup \{r'_1, ..., r'_n\}$.

As an example of rule elimination we are considering a technique by which every rule of the form

$$a_1 \vee ... \vee a_i \vee ... \vee a_k \leftarrow a_{k+1}, ..., a_i, ..., a_m, \text{not } a_{m+1}, ..., \text{not } a_n.$$

can be eliminated from a program Π. The atom a_i occurs in the head and unnegated in the body. The simplest example is the following rule r:

$$s(X) \leftarrow s(X).$$

For every program Π it holds that Π is strongly equivalent/uniformly equivalent to $\Pi - \{r\}$.

A technique for rule transformation we will study transforms a rule

$$a(X) \vee b(Y) \leftarrow c(X, Y)$$

into rules

$$a(X) \leftarrow c(X, Y), \text{not } b(Y)$$
$$b(Y) \leftarrow c(X, Y), \text{not } a(X)$$

The new rules do not have any disjunction in the head.

The most general way of program simplification is testing strong/uniform equivalence. Rule elimination and transformation are special cases of it. But testing strong equivalence of disjunctive datalog programs with negation is complete for coNEXPTIME and therefore provably intractable. Even worse, testing uniform equivalence is undecidable. This is the reason why we concentrate on special techniques for rule elimination and transformation besides an implementation of a strong equivalence test. Together with implementations of some techniques we provide a first prototype of a toolbox for program simplification and optimization of disjunctive datalog programs with negation.

In the literature, techniques for simplifying propositional datalog programs under the stable model semantics were already studied. But for practice, datalog programs with variables are more important. In this work, we will generalize techniques from the propositional to the non-ground case which were studied by Eiter et al. [EFTW04] and Lin and Chen [LC05]. Our main results are briefly summarized as follows:

- We introduce the generalized techniques which we named G-TAUT, G-CONTRA, G-RED$^-$, G-NONMIN, G-S-IMPL, G-SUBS, and G-LSH.
- We show that the generalized techniques preserve strong equivalence and/or uniform equivalence.
- We analyze the complexity of the applicability of the techniques.
- We implement the most important techniques and a test for deciding strong equivalence in general.

To see the implications of our work consider for example program optimization. There are two kinds of optimizations: online and offline optimization. Online means that the program might be updated at runtime. The algorithms for online optimization have to be efficient. In the general case such algorithms do not exist. Therefore we have to study special techniques for program simplification and optimization. Some of the studied techniques are indeed efficiently computable and hence can be used for online optimization. Others are not. For example, we show that testing applicability of G-LSH is PSPACE complete. But still this is an interesting technique since if G-LSH can be applied a program is produced where the problem of deciding satisfiability is complete for NEXPTIME instead of NEXPTIME$^{\mathrm{NP}}$ which is the complexity of the general case. Moreover, we

found a for practice important special case of G-LSH which is in NL.

The overall background of this thesis is Answer Set Programming (ASP), an area of logic programming receiving growing interest within the last years. Answer set programs and what we called here disjunctive datalog programs with negation are essentially the same. From a theoretical point of view ASP is a quite general approach to declarative problem solving because of its high expressive power. For example, problems complete for NP may be expressed easily in ASP. And the evaluation complexity of answer set programs has exponential time lower bounds. This is also a reason why program simplification and optimization are important. In practice, ASP has found its way into many different areas of applications: Planning, Verification and Configuration, Multi-Agent Systems, Security and Crypto-Analysis (e.g. verification of cryptographic protocols), Diagnostic Systems and Inconsistency Management. The basis for successful applications are systems like DLV[1] [D+01, L+05], Smodels[2] [NSS00, NSS02], NoMoRe[3] [AKL01, LAK02], ASSAT[4] [LZ02, LZ03], and Cmodels[5] [Lierler05]. All of them can be found on the web currently.

The outline of this thesis is as follows. The preliminaries, i.e., exact definitions and some theorems, are provided in Chapter 2. In particular we define disjunctive datalog programs with negation, equivalence of programs, provide some examples, shortly introduce computational complexity and list the major complexity results in answer set programming. In Chapter 3 we study rule elimination and in Chapter 4 rule transformation. Both chapters have the same structure. First we introduce the techniques and prove that they preserve strong and/or uniform equivalence. Then we study their complexity. Chapter 5 is devoted to implementations. Algorithms for testing strong equivalence and the most important techniques are introduced. We also address implementation issues. Finally we discuss further work in Chapter 6 and summarize our results in Chapter 7.

Parts of this thesis on the theory and practice of program simplification and optimization of non-ground disjunctive datalog programs with negation have already been accepted for publication [E+06, EFT05].

[1] http://www.dlvsystem.com
[2] http://www.tcs.hut.fi/Software/smodels
[3] http://www.cs.uni-potsdam.de/ linke/nomore/
[4] http://assat.cs.ust.hk
[5] http://www.cs.utexas.edu/users/tag/cmodels.html

Chapter 2

Preliminaries

This chapter provides the necessary background for our results. For an overview of Answer Set Programming see [Baral02, Niemelae99, MT99, GL02, FL05]. Some textbooks on computational complexity are [Pap94, Sipser97]. Further material about complexity classes can be found in [GJ79, Johnson90].

2.1 Disjunctive Datalog Programs

We fix a language $\mathcal{L} = \langle \mathcal{D}, \mathcal{R} \rangle$ where $\mathcal{D}$ is a countable infinite set of constants, the *domain*, and for every arity $k \in \mathbb{N} = \{0, 1, 2, ...\}$ there are countable infinite many relational symbols in $\mathcal{R}$, i.e., the set $\mathcal{R}^k$ of all relational symbols of arity k is countable infinite. Note that we also allow relational symbols of arity 0. We usually omit the parentheses in this case. There are no function symbols. An atom of arity k has the form

$$p(x_1, ..., x_k)$$

where x_i is a variable or a constant. We are using upper letters for variables, lower letters for relational symbols, and lower letters or numbers for constants, e.g., $p(X, 1)$, $q(1, c)$, $r(U, V, W)$ are atoms with variables U, V, W, X and constants $c, 1$.

Let a_i denote an atom. A *disjunctive datalog program with negation* (program for short) is a finite set of *rules* where each rule has the form

$$a_1 \vee ... \vee a_k \leftarrow a_{k+1}, ..., a_m, \text{not } a_{m+1}, ..., \text{not } a_n \tag{2.1}$$

and the *safety requirement* holds, i.e., every variable in $a_1, ..., a_k, a_{m+1}, ..., a_n$ occurs in $a_{k+1}, ..., a_m$.

For a rule r of the form (2.1) we define

- the *head* of r as $H(r) := \{a_1, ..., a_k\}$,
- the *body* of r as $B(r) := \{a_{k+1}, ..., a_m, \text{not } a_{m+1}, ..., \text{not } a_n\}$,
- the *positive body* of r as $B^+(r) := \{a_{k+1}, ..., a_m\}$, and
- the *negative body* of r as $B^-(r) := \{a_{m+1}, ..., a_n\}$.

A *constraint* is a rule with an empty head, and a *disjunctive fact* is a rule with an empty body. A program may contain constraints or disjunctive facts. A (non-disjunctive) *fact* is a rule with an empty body and without disjunction. We usually omit the symbol "$\leftarrow$" for facts. If the head consists of at most one atom then the rule is called *normal*; if the negative body is empty then the rule is called *positive*; and if the rule is normal and positive it is called *Horn*. A program is called *normal/positive/Horn* if every rule is normal/positive/Horn.

The *base* $B_{\mathcal{L}} = B_{\mathcal{R},\mathcal{D}}$ of a language $\mathcal{L}$ is the set of all ground atoms – i.e. variable-free atoms – build from $\mathcal{R}$ and $\mathcal{D}$. An *interpretation* in a language $\mathcal{L}$ is a subset of $B_{\mathcal{L}}$. For a program Π denote by $\mathcal{L}(\Pi) = \langle \mathcal{D}_\Pi, \mathcal{R}_\Pi \rangle$ the language of Π and by $\max_{ar}(\Pi)$ denote the maximal arity of relational symbols in $\mathcal{R}_\Pi$. The set $\mathcal{R}_\Pi$ contains exactly the relational symbols in Π, and the set $\mathcal{D}_\Pi$, which is called the *active domain* of Π, contains exactly the constants in Π.

Example 1. Let Π be

$$\begin{aligned} e(1,2) &\leftarrow \\ e(1,3) &\leftarrow \\ e(2,3) \vee e(3,2) &\leftarrow \\ v(X) &\leftarrow e(X,Y) \\ v(Y) &\leftarrow e(X,Y) \end{aligned}$$

The active domain is $\mathcal{D}_\Pi = \{1,2,3\}$, and $\mathcal{R}_\Pi = \{e, v\}$. The maximal arity is $\max_{ar}(\Pi) = 2$. □

Let r be a rule, $C \subseteq \mathcal{D}$ and $\vartheta : V_r \to C$ a *ground substitution* where V_r is the set of all variables in r. Then $r\vartheta$ denotes the result of the substitution, called the *instantiation* of r. Let $\mathrm{Gr}(\Pi, C) := \{r\vartheta : r \in \Pi, \vartheta : V_r \to C\}$ and for some fixed $c \in \mathcal{D}$

$$\mathrm{Gr}(\Pi) := \begin{cases} \mathrm{Gr}(\Pi, \mathcal{D}_\Pi) \text{ if } \mathcal{D}_\Pi \neq \{\} \\ \mathrm{Gr}(\Pi, \{c\}) \end{cases}$$

be the *grounding* of Π. An interpretation M is a *model* of Π, symbolically $M \models \Pi$, iff for all $r \in \mathrm{Gr}(\Pi)$:

$$\text{if } B^+(r) \subseteq M \text{ and } B^-(r) \cap M = \{\} \text{ then } H(r) \cap M \neq \{\}.$$

For an interpretation I of Π define

$$\Pi^I := \{H(r) \leftarrow B^+(r) : B^-(r) \cap I = \{\}, r \in \mathrm{Gr}(\Pi)\}$$

is the *reduct* of Π relative to I. A *stable model* of Π is a subset-minimal model of Π^I. Let $\mathrm{SM}(\Pi)$ denote the set of stable models of Π.

From this definition of a (stable) model it follows that only interpretations I with $I \subseteq B_{\mathcal{L}(\Pi)}$ are relevant. Note, that $B_{\mathcal{L}(\Pi)}$ is finite.

In what follows, we illustrate these basic notions on some examples.

Example 2. The grounding for the program

$$\begin{aligned} e(0,1) \\ e(Y,X) &\leftarrow e(X,Y) \end{aligned}$$

is

$$\begin{aligned} e(0,1) \\ e(0,0) &\leftarrow e(0,0) \\ e(1,0) &\leftarrow e(0,1) \\ e(0,1) &\leftarrow e(1,0) \\ e(1,1) &\leftarrow e(1,1) \end{aligned}$$

The only stable model is $\{e(0,1), e(1,0)\}$. □

Example 3. The program

$$\begin{aligned} e(1,2) \\ e(1,3) \\ e(2,3) \vee e(3,2) \\ v(X) \leftarrow e(X,Y) \\ v(Y) \leftarrow e(X,Y) \end{aligned}$$

has the stable models

$$\begin{aligned} I &:= \{e(1,2), e(1,3), e(2,3), v(1), v(2), v(3)\}, \\ J &:= \{e(1,2), e(1,3), e(3,2), v(1), v(2), v(3)\}. \end{aligned}$$

Because of the minimality of I and J neither $I \cup J$ nor any $K \supseteq I, J$ is a stable model of Π. For instance, $\{e(1,2), e(1,3), e(2,3), v(1), v(2), v(3), e(3,3)\}$ is not a stable model of Π □

Example 4. The program Π

$$\begin{aligned} a \\ c \leftarrow a, \text{not } b \\ b \leftarrow a, \text{not } c \end{aligned}$$

has the reduct $\Pi^I = \{a; b \leftarrow a\}$ relative to $I = \{a, b\}$ and the reduct $\Pi^J = \{a; c \leftarrow a\}$ relative to $J = \{a, c\}$. Since I is a subset-minimal model of Π^I and J a subset-minimal model of Π^J both are stable models of Π. But the classical model $K := \{a, b, c\}$ is not a stable model of Π because it is not a subset-minimal model of $\Pi^K = \{a\}$. □

2.2 Propositional Programs

A rule/program is called *propositional* iff it contains only relational symbols of arity 0. For instance, the program of Example 4 is propositional.

The relational symbols in $\mathcal{R}^0$ are also called *propositional atoms*. Let μ be a bijective mapping of ground atoms over $\mathcal{L}$ to propositional atoms from $\mathcal{R}^0$. Extend μ to be a bijective mapping of ground programs over $\mathcal{L}$ to

propositional programs over $\mathcal{R}^0$.

Example 5. For instance, μ assigns to $\{a(1) \leftarrow b(3), c(7); b(3) \leftarrow b(5)\}$ the propositional program $\{p \leftarrow q, r; q \leftarrow s\}$ where $\mu(a(1)) = p$, $\mu(b(3)) = q$, $\mu(c(7)) = r$, and $\mu(b(5)) = s$. □

2.3 Equivalences

Two programs P and Q are (*ordinary*) *equivalent*, symbolically $P \equiv Q$, iff they have the same stable models. They are *strongly equivalent*, symbolically $P \equiv_s Q$, iff for every program R it holds that $P \cup R \equiv Q \cup R$, and they are *uniformly equivalent*, symbolically $P \equiv_u Q$, iff for every finite set F of facts $P \cup F \equiv Q \cup F$.

The following proposition is an immediate consequence of these definitions.

Proposition 1. Let P and Q be programs. If $P \equiv_s Q$ then $P \equiv_u Q$.

For our purposes, we note that this proposition implies that every technique which preserves strong equivalence also preservers uniform equivalence. We will make use this result throughout the thesis without mentioning it.

Strong equivalence has a model-theoretic characterization. We need the following definition.

Definition 1. Let Π be a program and $C \subseteq \mathcal{D}$. An *SE-model* of Π is a pair $(X, Y)_C$ of interpretations $X, Y \subseteq B_{\mathcal{R},C}$ such that $X \subseteq Y$, $Y \models \mathrm{Gr}(\Pi, C)$, and $X \models \mathrm{Gr}(\Pi, C)^Y$, symbolically $(X, Y)_C \models_s \Pi$. □

$\mathrm{SE}_C(\Pi) := \{(X, Y)_C : X \subseteq Y \subseteq B_{\mathcal{R},C}, (X, Y)_C \models_s \Pi\}$.

$\mathrm{SE}(\Pi) := \bigcup_{C \subseteq \mathcal{D}} \mathrm{SE}_C(\Pi)$.

Instead of arbitrary sets C we may consider only finite sets $C \subseteq_{fin} \mathcal{D}$.

$\mathrm{SE}^{fin}(\Pi) := \bigcup_{C \subseteq_{fin} \mathcal{D}} \mathrm{SE}_C(\Pi)$.

Now we can state the model-theoretic characterization of strong equivalence. The theorem is used for example in proving that testing strong equivalence of two programs is decidable. We will use it later for showing that the techniques we study preserve strong equivalence.

Proposition 2 ([EFTW05]). Let P and Q be programs. Then, $P \equiv_s Q$ iff $\mathrm{SE}(P) = \mathrm{SE}(Q)$.

Inspecting the proof of this proposition in [EFTW05] we get the following more general proposition we will use later on.

Proposition 3. Let P and Q be programs. Then, $P \equiv_s Q$ iff $\mathrm{SE}^{fin}(P) = \mathrm{SE}^{fin}(Q)$.

Example 6. The programs

$$h(X) \leftarrow a(X)$$
$$t(X) \leftarrow h(X)$$
$$a(X) \leftarrow t(X)$$
$$a(X) \leftarrow h(X)$$

and

$$h(X) \leftarrow a(X)$$
$$t(X) \leftarrow h(X)$$
$$a(X) \leftarrow t(X)$$

are strongly equivalent. Eliminating the rule $a(X) \leftarrow h(X)$ from the first program does not change the SE-models. The first program states that $a \subseteq h$, $h \subseteq t$, $t \subseteq a$, and hence $a = h = t$. Therefore, the last rule which states $t \subseteq h$ can be eliminated. □

For the purpose of lifting we will need the propositional counterparts of these notions.

Two propositional programs P and Q are *propositional equivalent*, symbolically $P \equiv^{prop} Q$, iff they have the same propositional stable models. They are *propositional strongly equivalent*, symbolically $P \equiv_s^{prop} Q$, iff for every propositional program R it holds that $P \cup R \equiv^{prop} Q \cup R$, and they are *propositional uniformly equivalent*, symbolically $P \equiv_u^{prop} Q$, iff for every finite set F of propositional facts $P \cup F \equiv^{prop} Q \cup F$.

Definition 2. Let Π be a propositional program. A *propositional SE-model* of Π is a pair (X, Y) of interpretations $X, Y \subseteq \mathcal{R}^0$ such that $X \subseteq Y$, $Y \models \Pi$, and $X \models \Pi^Y$, symbolically $(X, Y) \models_s^{prop} \Pi$. □

$$\mathrm{SE}^{prop}(\Pi) := \{(X, Y) : X \subseteq Y \subseteq \mathcal{R}^0, (X, Y) \models_s^{prop} \Pi\}.$$

Proposition 4 ([Turner01, Turner03]). Let P and Q be propositional programs. Then $P \equiv_s^{prop} Q$ iff $\mathrm{SE}^{prop}(P) = \mathrm{SE}^{prop}(Q)$.

If P and Q are propositional programs then $P \equiv_s Q$ iff $P \equiv_s^{prop} Q$. Another connection between strong equivalence and propositional strong equivalence is given in the following lemma. We will use it later on in the correctness proofs of the techniques. It involves the mapping μ between ground atoms and propositional atoms we defined in the previous section.

Lemma 1. Let P and Q be a programs and $C \subseteq \mathcal{D}$. If $\mu(\mathrm{Gr}(P, C)) \equiv_s^{prop} \mu(\mathrm{Gr}(Q, C))$ then $\mathrm{SE}_C(P) = \mathrm{SE}_C(Q)$.

Proof. Let $(X, Y)_C \in \mathrm{SE}_C(P)$. The following implications are easy consequences of the involved definitions: $X \subseteq Y \subseteq B_{\mathcal{R},C}$ implies $\mu(X) \subseteq \mu(Y) \subseteq \mathcal{R}^0$, $Y \models \mathrm{Gr}(P, C)$ implies $\mu(Y) \models \mu(\mathrm{Gr}(P, C))$, and $X \models \mathrm{Gr}(P, C)^Y$ implies $\mu(X) \models \mu(\mathrm{Gr}(P, C))^{\mu(Y)}$. Hence $(\mu(X), \mu(Y)) \in \mathrm{SE}^{prop}(\mu(\mathrm{Gr}(P, C)))$ and $(\mu(X), \mu(Y)) \in \mathrm{SE}^{prop}(\mu(\mathrm{Gr}(Q, C)))$. The following implications also hold: $\mu(Y) \models \mu(\mathrm{Gr}(Q, C))$ implies $Y \models \mathrm{Gr}(Q, C)$, and $\mu(X) \models \mu(\mathrm{Gr}(Q, C))^{\mu(Y)}$ implies $X \models \mathrm{Gr}(Q, C)^Y$. Therefore $(X, Y)_C \in \mathrm{SE}_C(Q)$. One shows that $(X, Y)_C \in \mathrm{SE}_C(P)$ for $(X, Y)_C \in \mathrm{SE}_C(Q)$ in the same way. □

2.4 Head-cycle Free Programs

Let Π be a program and A be the set of all atoms in $\mathrm{Gr}(\Pi)$. The *dependency graph* $D(\mathrm{Gr}(\Pi))$ of Π has exactly the vertices A and contains an edge $e = (a, b)$ iff there exists a rule $r \in \mathrm{Gr}(\Pi)$ such that $a \in H(r)$ and $b \in B^+(r) \cup B^-(r)$, moreover, e is marked with $+$ if $b \in B^+(r)$ and e is marked with $-$ if $b \in B^-(r)$. Denote by $D^+(\mathrm{Gr}(\Pi))$ the graph restricted to edges in $D(\mathrm{Gr}(\Pi))$ which have only the mark $+$. An atom a *positively depends* on an atom b iff there exists a path from a to b in $D^+(\mathrm{Gr}(\Pi))$. A rule $r \in \mathrm{Gr}(\Pi)$ is *head-cycle free* iff no distinct atoms $a, b \in H(r)$ mutually positively depend on each other. A program Π is *head-cycle free* iff every rule $r \in \mathrm{Gr}(\Pi)$ is head-cycle free.

Example 7. Consider the program

$$\begin{aligned} a \vee d &\leftarrow b, c \\ b \vee c &\leftarrow a \\ a \vee e &\leftarrow d \end{aligned}$$

The third rule is head-cycle free in the program. The first rule is not head-cycle free because (d, b, a, d) is a cycle in the positive dependency graph and the second rule is not head-cycle free because of the cycle (b, a, c, a, b). □

Example 8. The program

$$\begin{aligned} r(X) \vee u(X) &\leftarrow s(X) \\ s(X) &\leftarrow t(X, Y) \\ t(X, 0) &\leftarrow u(0) \\ u(1) &\leftarrow r(X) \end{aligned}$$

is head-cycle free. But the program

$$\begin{aligned} r(X) \vee u(X) &\leftarrow s(X) \\ s(X) &\leftarrow t(X, Y) \\ t(X, 0) &\leftarrow u(0) \\ u(0) &\leftarrow r(X) \end{aligned}$$

is not. The head-cycle in the latter program is $(r(0), s(0), t(0, 0), u(0), r(0))$. □

2.5 Computational Complexity

In this section we present all notions and results we need from complexity theory. We also give an overview of complexity results concerning disjunctive datalog programs with negation.

Let Σ be a finite alphabet, Σ^* be the set of all finite words over Σ, and denote by $|x|$ the number of symbols in the string $x \in \Sigma^*$. A language $L \subseteq \Sigma^*$ is in P iff there exists a polynomial function $p : \mathbb{N} \to \mathbb{N}$ and a Turing-Machine M which accepts L and for all $x \in L$: $t_M(x) \leq p(|x|)$. The function $t_M(x)$ denotes the time complexity of M for the input x, i.e., the number of executed steps. We also consider space complexity. Denote by $s_M(x)$ the space complexity of M for the input x, i.e., the number of used tape cells. A language $L \subseteq \Sigma^*$ is in PSPACE iff there exists a polynomial function $p : \mathbb{N} \to \mathbb{N}$ and a Turing-Machine M which accepts L and for all $x \in L$: $s_M(x) \leq p(|x|)$, and L is in the class L iff for all $x \in L$: $s_M(x) \leq \lg(p(|x|))$.

The classes L, P, and PSPACE have the nondeterministic counterparts NL, NP, and NPSPACE. It holds that PSPACE = NPSPACE and coNL = NL where coNL is the complement class of NL. The relationship of these classes is: L $\subseteq$ NL $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE. Denote by FL the class of all functions $f : \Sigma^* \to \Sigma^*$ which are computable in logarithmic space. A language L_1 is *reducible* to a language L_2, symbolically $L_1 \leq_m^{\mathrm{L}} L_2$, iff there exists $f \in$ FL such that for all $x \in \Sigma^*$: $x \in L_1$ iff $f(x) \in L_2$. A language L is *complete* for a complexity class $\mathcal{C}$ iff $L \in \mathcal{C}$ and for all $L' \in \mathcal{C}$: $L' \leq_m^{\mathrm{L}} L$. We will also use polynomial time reductions. An introductory treatment of these notions can be found in the book [Sipser97] of Sipser. There one also find a more accurate definition of an oracle Turing machine we provide here. Informally, an oracle Turing machine M^L for a language (oracle) L can decide L with unit cost. Oracle complexity classes are defined in the usual way. See also [Pap94, S76].

The following two problems will be used later on. A digraph G is a pair (V, E) of a finite set V of vertices and of a set E of directed edges, i.e. $E \subseteq V \times V$. Let $s, t \in V$. We say that t is *reachable* from s in G iff there

exists a path from s to t in G. The problem of deciding if t is reachable from s in G is complete for NL. A graph G is a pair (V, E) of a finite set V of vertices and of a set E of undirected edges, i.e., E consists of two element subsets of V. A *3-coloring* of a graph G is a function $\chi : V \to \{1, 2, 3\}$ such that for all $\{x, y\} \in E$: $\chi(x) \neq \chi(y)$. Deciding if there exists a 3-coloring of a graph G is complete for NP.

Now we give a short overview of complexity results concerning disjunctive datalog programs with negation. We will not need these results for our proofs but they should help in getting a better understanding of the computational cost needed to solve problems in the area of Answer Set Programming.

First, we consider the problem to decide if a program of a particular form (Horn, Normal, General) has a stable model. The following results are well known. See for example [EGM97], [EF+04], [DE+01].

	propositional	non-ground
Horn	P	EXPTIME
Normal	NP	NEXPTIME
General	NP^{NP}	$\text{NEXPTIME}^{\text{NP}}$

EXPTIME is the set of all problems solvable in exponential time. All the problems are indeed complete for the complexity classes. Next we consider the problem to decide strong/uniform equivalence.

	propositional	non-ground
SE	coNP [PTW01, Lin02]	coNEXPTIME [EFTW05]
UE	coNP^{NP} [EF03]	undecidable [EFTW05]

All the problems with the exception to decide uniform equivalence of (non-ground) programs are complete for the complexity classes.

We are also interested in reasoning problems, especially brave reasoning. For a negated or an unnegated ground atom b and a program Π the problem is to decide if there exists an $I \in \text{SM}(\Pi)$ such that $b \in I$. We only need the brave reasoning problem for Horn programs. In the propositional case this problem is complete for P and in the non-ground case complete for EXPTIME.

2.6 Further Examples

In this section we present further examples to illustrate how, for instance, Answer Set Programming is applied for solving problems over graphs.

Example 9. 3-Coloring.

$$\begin{aligned} e(Y,X) &\leftarrow e(X,Y) \\ r(X) \vee g(X) \vee b(X) &\leftarrow e(X,Z) \\ &\leftarrow g(X), g(Y), e(X,Y) \\ &\leftarrow b(X), b(Y), e(X,Y) \\ &\leftarrow r(X), r(Y), e(X,Y) \end{aligned}$$

Adding a set E of facts of the form $e(c,d)$ yields a program which describes by its stable models the admissible 3-colorings of the graph represented by E. □

Example 10. Transitive closure. The programs

$$\begin{aligned} t(X,Y) &\leftarrow e(X,Y) \\ t(X,Z) &\leftarrow t(X,Y), t(Y,Z) \end{aligned}$$

and

$$\begin{aligned} t(X,Y) &\leftarrow e(X,Y) \\ t(X,Z) &\leftarrow e(X,Y), t(Y,Z) \end{aligned}$$

both compute the transitive closure t of e. They are ordinary equivalent and remain ordinary equivalent if any set E of facts $e(c,d)$ is added. But they are neither strongly equivalent nor uniformly equivalent. Adding $\{t(1,2), t(2,3)\}$ to both programs yields non-equivalent programs. □

Example 11. Here are two programs which are strongly equivalent and contain negation.

Program 1:

$$a(X) \leftarrow \text{not } b(X)$$
$$a(X) \leftarrow c(X)$$
$$c(X) \leftarrow a(X)$$
$$c(X) \leftarrow \text{not } b(X)$$

Program 2:

$$a(X) \leftarrow \text{not } b(X)$$
$$a(X) \leftarrow c(X)$$
$$c(X) \leftarrow a(X)$$

Removing the last rule of the first programs does not change the SE-models.
□

Chapter 3

Rule Elimination

In this chapter we study rule elimination. First, we recall techniques for rule elimination in the propositional case which are subsequently generalized to the non-ground case. These generalizations were first pointed out in an informal way by Eiter et al. [E+04].

Definition 3. A rule r is an *s-implication* of a rule r', symbolically $r' \lhd r$, iff there exists $A \subseteq B^-(r)$ such that $H(r') \subseteq H(r) \cup A$, $B^-(r') \subseteq B^-(r) - A$, $B^+(r') \subseteq B^+(r)$. □

Proposition 5 ([ONA01, EFTW04]). Let Π be a propositional program and $r, r' \in \Pi$. $\Pi \equiv_s^{prop} \Pi - \{r\}$ if one of the following conditions hold

- TAUT(r): $H(r) \cap B^+(r) \neq \{\}$
- CONTRA(r): $B^+(r) \cap B^-(r) \neq \{\}$
- RED$^-$(r, r'): $H(r') \subseteq B^-(r), B(r') = \{\}$
- NONMIN(r, r'): $H(r) \subseteq H(r'), B(r) \subseteq B(r')$
- S-IMP(r, r'): $r' \lhd r$

In the previous proposition we named the conditions. We often say that TAUT is the technique where TAUT(r) has to hold such that we can eliminate r. And instead of saying that TAUT(r) does not hold we use the notation $\overline{\text{TAUT}}(r)$. The rules TAUT, CONTRA, RED$^-$, and NONMIN are due to Brass and Dix [BD99] and the rule S-IMP is originally due to Wang

and Zhou [WZ05].

Example 12. The rules $a \leftarrow a, b$ and $c \leftarrow a, \text{not } a$ can be eliminated from the program Π

$$\begin{aligned} a &\leftarrow a, b \\ a \vee c &\leftarrow b \\ c &\leftarrow a, \text{not } a \end{aligned}$$

by applying TAUT, and respectively CONTRA. It holds that $\Pi \equiv_s \{a \vee c \leftarrow b\}$. □

If NONMIN(r, r') holds then also S-IMP(r, r'). The condition SUBS(r, r') in the following proposition is even more general in the sense that it implies NONMIN(r, r'). We note this in the next but one proposition.

Proposition 6 ([LC05]). Let Π be a propositional program and $r, r' \in \Pi$. $\Pi \equiv_s^{prop} \Pi - \{r\}$ if the following condition holds

- SUBS(r, r'): $B^+(r') \subseteq B^+(r)$, $B^-(r') \subseteq B^-(r)$, $H(r') \subseteq H(r) \cup B^-(r)$.

Proposition 7. Let r, r' be propositional rules. The following implications hold:

- NONMIN$(r, r') \Rightarrow$ S-IMP$(r, r') \Rightarrow$ SUBS(r, r'),
- RED$^-(r, r') \Rightarrow$ S-IMP$(r, r') \Rightarrow$ SUBS(r, r').

In what follows we provide generalizations of these conditions and show that the according techniques preserve strong equivalence. Recall that by Proposition 1, this immediately implies that the techniques also preserve uniform equivalence.

3.1 G-TAUT and G-CONTRA

We start with the generalization of TAUT and CONTRA to the non-ground case.

Definition 4. Let r be a rule. G-TAUT(r) iff $H(r) \cap B^+(r) \neq \{\}$. □

Definition 5. Let r be a rule. G-CONTRA(r) iff $B^+(r) \cap B^-(r) \neq \{\}$. □

The conditions in the non-ground case are the same as in the propositional case. We have to show that these techniques remain correct, i.e., if G-TAUT(r) or G-CONTRA(r) holds then r can be eliminated from any program such that the resulting program remains strongly equivalent.

Theorem 1. (Correctness) Let Π be a program and $r \in \Pi$. Then, $\Pi \equiv_s \Pi - \{r\}$ if

1. G-TAUT(r) or
2. G-CONTRA(r).

Proof. 1) Because of Proposition 3 we have to show $\mathrm{SE}_C(\Pi) = \mathrm{SE}_C(\Pi - \{r\})$ for every $C \subseteq_{fin} \mathcal{D}$. Let $C \subseteq_{fin} \mathcal{D}$. We only need to show that $\mu(\mathrm{Gr}(\Pi, C)) \equiv_s^{prop} \mu(\mathrm{Gr}(\Pi - \{r\}, C))$ (cf. Lemma 1).

Define

$$\Delta := \mathrm{Gr}(\Pi, C) - \mathrm{Gr}(\Pi - \{r\}, C).$$

We are showing $\mu(\mathrm{Gr}(\Pi, C)) \equiv_s^{prop} \mu(\mathrm{Gr}(\Pi, C)) - \mu(S)$ by induction on the size of $S \subseteq \Delta$. The induction start $|S| = 0$ is clear. Assume that $\mu(\mathrm{Gr}(\Pi, C)) \equiv_s \mu(\mathrm{Gr}(\Pi, C)) - \mu(S)$ for $0 < |S| < |\Delta|$, and let $r' \in \Delta - S$. The rule r' is an instantiation of r, and because of G-TAUT(r) it holds that TAUT($\mu(r')$). Therefore by Proposition 5 $\mu(\mathrm{Gr}(\Pi, C)) - \mu(S) - \{r'\} \equiv_s^{prop} \mu(\mathrm{Gr}(\Pi, C)) - \mu(S) \equiv_s^{prop} \mu(\mathrm{Gr}(\Pi, C))$.

This proves $\mu(\mathrm{Gr}(\Pi, C)) \equiv_s \mu(\mathrm{Gr}(\Pi, C)) - \mu(\Delta)$ and hence $\mathrm{SE}_C(\Pi) = \mathrm{SE}_C(\Pi - \{r\})$.

2) Analogous. □

Example 13. For example, $s(X) \vee t(X) \leftarrow s(X)$ is redundant because every instantiation $s(c) \vee t(c) \leftarrow s(c)$ is a tautology. $t(c)$ is not derived because of minimality. Another example is $\leftarrow s(X), \text{not } s(X)$. Every instantiation $\leftarrow s(c), \text{not } s(c)$ is a contradiction. □

There is another way to generalize the techniques from the propositional case:

Definition 6. Let r be a rule. G-TAUT$^*(r)$ iff for all $\vartheta : V_r \rightarrow \mathcal{D}$: $H(r\vartheta) \cap B^+(r\vartheta) \neq \{\}$. □

Definition 7. Let r be a rule. G-CONTRA$^*(r)$ iff for all $\vartheta : V_r \rightarrow \mathcal{D}$: $B^+(r\vartheta) \cap B^-(r\vartheta) \neq \{\}$. □

Obviously G-TAUT(r) implies G-TAUT$^*(r)$, and G-CONTRA(r) implies G-CONTRA$^*(r)$, i.e., these generalization might be even more general than the previous ones. But it shows up that they are equivalent.

Theorem 2. Let Π be a program and $r \in \Pi$.

1. G-TAUT$^*(r)$ iff G-TAUT(r).
2. G-CONTRA$^*(r)$ iff G-CONTRA(r).

Proof. 1) $\Rightarrow$) By contraposition. $\overline{\text{G-TAUT}}(r)$ iff $H(r) \cap B^+(r) = \{\}$. Define $\vartheta(X_i) := c_i$ where $X_1, ..., X_n$ are the variables in r and $c_1, ..., c_n$ are new and distinct constants; ϑ is a ground substitution and of the form $V_r \rightarrow \{c_1, ..., c_n\}$. Note, that the domain is infinite. Let $a \in H(r), b \in B^+(r)$. If a and b have different relational symbols then $a\vartheta \neq b\vartheta$. Otherwise, they have at least one different variable at some place because of $H(r) \cap B^+(r) = \{\}$ and again $a\vartheta \neq b\vartheta$. Hence, $H(r\vartheta) \cap B^+(r\vartheta) = \{\}$ which implies $\overline{\text{G-TAUT}}^*(r)$.

$\Leftarrow$) Let $a \in H(r) \cap B^+(r)$. For all $\vartheta : V_r \rightarrow \mathcal{D}$: $a\vartheta \in H(r\vartheta) \cap B^+(r\vartheta)$.

2) Analogous. □

3.2 G-RED$^-$

Next, we provide the generalization of RED$^-$.

Definition 8. Let r and s be rules. G-RED$^-(r, s)$ iff $H(s) \subseteq B^-(r)$ and $B(s) = \{\}$. □

No disjunctive fact contains a variable because of the safety requirement. This explains why G-RED$^-$ and RED$^-$ are the same.

Theorem 3. (Correctness) Let Π be a program and $r, s \in \Pi$. If G-RED$^-(r, s)$ then $\Pi \equiv_s \Pi - \{r\}$.

Proof. This proof is similar to that of Theorem 1. We only need to prove that $\mu(\mathrm{Gr}(\Pi, C)) \equiv_s^{prop} \mu(\mathrm{Gr}(\Pi - \{r\}, C))$ for any $C \subseteq_{fin} \mathcal{D}$ because of Proposition 3 and Lemma 1.

Define

$$\Delta := \mathrm{Gr}(\Pi, C) - \mathrm{Gr}(\Pi - \{r\}, C).$$

We are showing $\mu(\mathrm{Gr}(\Pi, C)) \equiv_s^{prop} \mu(\mathrm{Gr}(\Pi, C)) - \mu(S)$ again by induction on the size of $S \subseteq \Delta$. The induction start $|S| = 0$ is clear. Assume that $\mu(\mathrm{Gr}(\Pi, C)) \equiv_s \mu(\mathrm{Gr}(\Pi, C)) - \mu(S)$ for $0 < |S| < |\Delta|$, and let $r' \in \Delta - S$. The rule r' is an instantiation of r, and because of G-RED$^-(r, s)$ it holds that $r' = r$ since s has to be ground because of the safety requirement and $H(s) = \{\}$. It then follows that RED$^-(\mu(r'), \mu(s))$ holds. Therefore by Proposition 5 $\mu(\mathrm{Gr}(\Pi, C)) - \mu(S) - \{r'\} \equiv_s^{prop} \mu(\mathrm{Gr}(\Pi, C)) - \mu(S) \equiv_s^{prop} \mu(\mathrm{Gr}(\Pi, C))$. This proves $\mu(\mathrm{Gr}(\Pi, C)) \equiv_s \mu(\mathrm{Gr}(\Pi, C)) - \mu(\Delta)$. □

3.3 G-NONMIN, G-S-IMPL, and G-SUBS

In this section we provide generalizations of NONMIN, S-IMPL, and SUBS.

Definition 9. Let r and s be rules. G-NONMIN(r, s) iff for all $C \subseteq_{fin} \mathcal{D}$ and $r' \in \mathrm{Gr}(r, C)$ there exists an $s' \in \mathrm{Gr}(s, C)$, such that $H(s') \subseteq H(r')$ and $B(s') \subseteq B(r')$. □

Definition 10. Let r and s be rules. G-S-IMPL(r, s) iff for all $C \subseteq_{fin} \mathcal{D}$ and $r' \in \mathrm{Gr}(r, C)$ there exists an $s' \in \mathrm{Gr}(s, C)$, such that $s' \lhd r'$. □

Definition 11. Let r and s be rules. G-SUBS(r, s) iff for all $C \subseteq_{fin} \mathcal{D}$ and $r' \in \mathrm{Gr}(r, C)$ there exists an $s' \in \mathrm{Gr}(s, C)$, such that $B^+(s') \subseteq B^+(r')$, $B^-(s') \subseteq B^-(r')$, and $H(s') \subseteq H(r') \cup B^-(r')$. □

Proposition 8. Let r and s be rules. G-SUBS(r, s) holds, whenever G-NONMIN(r, s) or G-S-IMPL(r, s) holds.

Theorem 4. (Correctness) Let Π be a program and $r, s \in \Pi$. Then, $\Pi \equiv_s \Pi - \{r\}$ if

1. G-NONMIN(r, s),
2. G-S-IMPL(r, s), or
3. G-SUBS(r, s).

Proof. 3) This proof is similar to that of Theorem 1. We only need to prove that $\mu(\mathrm{Gr}(\Pi, C)) \equiv_s^{prop} \mu(\mathrm{Gr}(\Pi - \{r\}, C))$ for any $C \subseteq_{fin} \mathcal{D}$ because of Proposition 3 and Lemma 1.

Define

$$\Delta := \mathrm{Gr}(\Pi, C) - \mathrm{Gr}(\Pi - \{r\}, C).$$

We are showing $\mu(\mathrm{Gr}(\Pi, C)) \equiv_s^{prop} \mu(\mathrm{Gr}(\Pi, C)) - \mu(S)$ again by induction on the size of $S \subseteq \Delta$. The induction start $|S| = 0$ is clear. Assume that $\mu(\mathrm{Gr}(\Pi, C)) \equiv_s \mu(\mathrm{Gr}(\Pi, C)) - \mu(S)$ for $0 < |S| < |\Delta|$, and let $r' \in \Delta - S$. The rule r' is an instantiation of r, and because of G-SUBS(r, s) it follows that there exists an s' such that SUBS$(\mu(r'), \mu(s'))$ holds. Therefore $\mu(\mathrm{Gr}(\Pi, C)) - \mu(S) - \{r'\} \equiv_s^{prop} \mu(\mathrm{Gr}(\Pi, C)) - \mu(S) \equiv_s^{prop} \mu(\mathrm{Gr}(\Pi, C))$. This proves $\mu(\mathrm{Gr}(\Pi, C)) \equiv_s \mu(\mathrm{Gr}(\Pi, C)) - \mu(\Delta)$.

1, 2) If G-NONMIN(r, s) or G-S-IMPL(r, s) holds then also G-SUBS(r, s) holds by Proposition 8 and hence $\Pi \equiv_s \Pi - \{r\}$. □

NONMIN(r, s) implies G-NONMIN(r, s), but the converse does not hold. Consider the following example.

Example 14. Let r be the rule

$$s(X_2) \leftarrow t(X_1, 0), t(0, X_2), t(X_2, X_2)$$

and s the rule

$$s(Y_3) \leftarrow t(Y_1, Y_2), t(Y_2, Y_3).$$

G-NONMIN(r, s) holds, but NONMIN(r, s) does not hold. But after substituting also NONMIN(r, s) does. Define $\vartheta := \{Y_1 \mapsto X_1, Y_2 \mapsto 0, Y_3 \mapsto X_2\}$ or $\vartheta := \{Y_1 \mapsto 0, Y_2 \mapsto X_2, Y_3 \mapsto X_2\}$. It holds that $H(s\vartheta) \subseteq H(r)$ and $B(s\vartheta) \subseteq B(r)$, i.e., NONMIN$(r\vartheta, s)$ holds. □

The previous example motivates the following definitions, providing an alternative generalization of NONMIN, S-IMP, and SUBS to the non-ground case.

Definition 12. Let r and s be rules. NONMIN$^*(r, s)$ iff there exists a substitution $\vartheta : V_s \rightarrow V_r \cup \mathcal{D}_r$, such that $H(s\vartheta) \subseteq H(r)$ and $B(s\vartheta) \subseteq B(r)$. □

Definition 13. Let r and s be rules. S-IMPL$^*(r, s)$ iff there exists a substitution $\vartheta : V_s \rightarrow V_r \cup \mathcal{D}_r$ and $A \subseteq B^-(r)$, such that $H(s\vartheta) \subseteq H(r) \cup A$, $B^-(s\vartheta) \subseteq B^-(r) - A$, and $B^+(s\vartheta) \subseteq B^+(r)$. □

Definition 14. Let r and s be rules. SUBS$^*(r, s)$ and there exists a substitution $\vartheta : V_s \rightarrow V_r \cup \mathcal{D}_r$, such that $B^+(s\vartheta) \subseteq B^+(r)$, $B^-(s\vartheta) \subseteq B^-(r)$, and $H(s\vartheta) \subseteq H(r) \cup B^-(r)$. □

Theorem 5. Let Π be a program and $r, s \in \Pi$. Then,

1. G-NONMIN(r, s) iff NONMIN$^*(r, s)$.
2. G-S-IMPL(r, s) iff S-IMPL$^*(r, s)$.
3. G-SUBS(r, s) iff SUBS$^*(r, s)$.

Proof. We prove (2). (1) and (3) are analogous.

$2, \Rightarrow$) Let $\{c_1, ..., c_m\} = \mathcal{D}_r \cup \mathcal{D}_s$ and $\{X_1, ..., X_n\} = V_r$. Since $\mathcal{D}$ is infinite there exists $C \subseteq \mathcal{D}$ such that $|C| \geq m + n$. Define $\vartheta_1(X_i) := d_i$

where $c_1, ..., c_m, d_1, ..., d_n$ are distinct and from C; ϑ_1 has the form $V_r \to C$. Let $r' = r\vartheta_1$. It follows from G-S-IMPL(r, s) that there exists $s' \in \mathrm{Gr}(s, C)$, i.e., $\exists \vartheta_2 : V_s \to C$ with $s' = s\vartheta_2$, and $A \subseteq B^-(r')$ such that $H(s') \subseteq H(r') \cup A, B^-(s') \subseteq B^-(r') - A, B^+(s') \subseteq B^+(r')$. Now, define

$$\vartheta(X) := \begin{cases} Y \text{ if } \vartheta_1(Y) = \vartheta_2(X) \\ \vartheta_2(X) \end{cases}$$

The substitution ϑ is well defined because if there exists Y with $\vartheta_1(Y) = \vartheta_2(X)$ then there exists exactly one since ϑ_1 is bijective. Because of the same reason and since variables are only substituted with new constants we can reverse the substitution for all atoms in A and get $E = A\vartheta_1^{-1} \subseteq B^-(r)$.

Claim: $H(s\vartheta) \subseteq H(r) \cup E$. For an atom a denote by $a^{(i)}$ the content of the i-th place of a. Let $b \in H(s)$. Then, there exists $a \in H(r) \cup E$ such that $a\vartheta_1 = b\vartheta_2 = p(c'_1, ..., c'_k)$ for a relational symbol p. Now, assume $a^{(i)}$ is a variable Y. Then $b^{(i)}$ can not be a constant because in the definition of ϑ_1 every variable is mapped to a different constant symbol not occurring in r or s, i.e., no variable is mapped to c_j for some j in this case. It follows that $b^{(i)}$ is a variable X and after the definition of ϑ it holds that $\vartheta(X) = Y$. Next, assume that $a^{(i)}$ is a constant c. If $b^{(i)}$ is also a constant c' then $c' = c$. Otherwise, let $b^{(i)}$ be the variable X. Since there exists no Y with $\vartheta_1(Y) = \vartheta_2(X)$ it holds that $\vartheta(X) = \vartheta_2(X) = c$. Hence, $H(s\vartheta) \subseteq H(r) \cup E$. The same argument shows $B^-(s\vartheta) \subseteq B^-(r) - A$ and $B^+(s\vartheta) \subseteq B^+(r)$. From this it is also clear that ϑ has the form $V_s \to V_r \cup \mathcal{D}_r$.

$2, \Leftarrow$) Let $C \subseteq \mathcal{D}$ and $r' \in \mathrm{Gr}(r, C)$. There exists a substitution $\vartheta_1 : V_r \to C$ such that $r' = r\vartheta_1$. Let ϑ and A be as in the definition of S-IMPL$^*(r, s)$. Define

$$\vartheta_2(X) := \begin{cases} \vartheta(X) \text{ if } \vartheta(X) \in \mathcal{D} \\ \vartheta_1(\vartheta(X)) \end{cases}$$

The substitution ϑ_2 is of the form $V_s \to C$, and therefore $s\vartheta_2 \in \mathrm{Gr}(s, C)$. It holds that $A\vartheta_1 \subseteq B^-(r')$, $H(s\vartheta_2) \subseteq H(r') \cup A\vartheta_1$, $B^-(s\vartheta_2) \subseteq B^-(r') - A\vartheta_1$, and $B^+(s\vartheta_2) \subseteq B^+(r')$, i.e., $s' \lhd r'$ for $s' = s\vartheta_2$. Hence, G-S-IMPL(r, s). □

The previous theorem shows that G-NONMIN, G-S-IMPL, and G-SUBS are decidable. In the next section we will analyze their complexity.

3.4 Complexity Analysis

G-TAUT, G-CONTRA, and G-RED$^-$ are obviously polynomial time decidable. For analyzing the complexity of deciding G-NONMIN, G-S-IMPL, and G-SUBS a special case of the subsumption problem is useful.

Definition 15. Let A and B be two sets of atoms without function symbols. A *subsumes* B iff there exists a substitution ϑ from the variables of A into the variables and constants of B such that $A\vartheta \subseteq B$. □

The following proposition is well known.

Proposition 9. $\{(A, B) : A \text{ subsumes } B\}$ is complete for NP.

To see this, consider a digraph G over the vertices $\{1, ..., n\}$. Define $B := \{e(r, g), e(g, r), e(r, b), e(b, r), e(g, b), e(b, g)\}$ and $e(X_i, X_j) \in A$ iff (i, j) is an edge in G. This establishes a reduction from the 3-Colorability problem.

Theorem 6. Deciding any of the sets

1. $\{(r, s) : \text{G-NONMIN}(r, s) \text{ holds}, r, s \text{ rules}\}$,
2. $\{(r, s) : \text{G-S-IMPL}(r, s) \text{ holds}, r, s \text{ rules}\}$, or
3. $\{(r, s) : \text{G-SUBS}(r, s) \text{ holds}, r, s \text{ rules}\}$

is complete for NP.

Proof. Let r and s be rules. We only need to consider NONMIN$^*(r, s)$, S-IMPL$^*(r, s)$, and S-SUBS$^*(r, s)$ because of Theorem 5. In every three cases membership can be decided by guessing a substitution $\vartheta : V_s \to V_r \cup \mathcal{D}_r$ and checking the respective conditions from Proposition 5. For the hardness proof assume that r and s are constraints with only positive atoms, i.e., $H(r) = H(s) = B^-(r) = B^-(s) = \{\}$. This special case is the subsumption problem which is NP-complete (cf. Proposition 9). □

Hence, in implementing these techniques we will consider G-SUBS(r, s) only because it is a generalization of G-NONMIN(r, s) and G-S-IMPL(r, s) with the same complexity to decide.

Chapter 4

Rule Transformation

In this chapter we study rule transformation. First, we recall a technique for rule transformation in the propositional case which is subsequently generalized to the non-ground case. This generalization was pointed out in an informal way by Eiter et al. [E+04], and stresses under which circumstances, disjunction can be eliminated.

Define for a rule r its *shift* as

$$r^{\rightarrow} := \begin{cases} \{a \leftarrow B(r), \text{not } (H(r) - \{a\}) : a \in H(r)\} \text{ if } |H(r)| > 1 \\ \{r\} \end{cases}$$

Definition 16. Let Π be a propositional program and $r \in \Pi$. Then, $\text{LSH}(\Pi, r)$ iff r is head-cycle free in Π and $|H(r)| > 1$. □

Proposition 10 ([EFTW04]). Let Π be a propositional program and $r \in \Pi$. If $\text{LSH}(\Pi, r)$ then $\Pi \equiv_u^{prop} \Pi - \{r\} \cup r^{\rightarrow}$.

The following proposition will be used in the proof of Theorem 7.

Proposition 11. Let Π be a propositional program and $r \in \Pi$. Then $\Pi \equiv^{prop} \Pi \cup r^{\rightarrow}$.

Proof. $\Rightarrow$) Let I be a stable model of Π. If $I \cap B^-(r) \neq \{\}$ then $I \cap B^-(i) \neq \{\}$ for every $i \in r^{\rightarrow}$. Let $I \cap B^-(r) = \{\}$. Since $I \models r^I$ it holds that

$I \cap H(r) \neq \{\}$. If $|I \cap H(r)| > 1$ then for every rule in $i \in r^{\rightarrow}$: $I \cap B^-(i) \neq \{\}$. If $|I \cap H(r)| = 1$ then exactly the rule $i \in r^{\rightarrow}$ with $a \in I \cap H(i)$ is in $\mathrm{Gr}(\Pi)^I$, and it is satisfied.

$\Leftarrow$) Let I be a stable model of $\Pi \cup r^{\rightarrow}$ and assume $I \cap B^-(r) = \{\}$. Because of the same reason as before there exists exactly one rule $i \in r^{\rightarrow}$ with $a \in I \cap H(i)$ which is in $\mathrm{Gr}(\Pi)^I$, and is satisfied. □

4.1 G-LSH

We generalize LSH to the non-ground case next.

Definition 17. Let Π be a program and $r \in \Pi$. G-LSH(Π, r) iff for all $C \subseteq_{fin} \mathcal{D}$ and $r' \in \mathrm{Gr}(r, C)$ it holds that r' is head-cycle free in $\mathrm{Gr}(\Pi, C)$ and $|H(r')| > 1$. □

Theorem 7. (Correctness) Let Π be a program and $r \in \Pi$. Then, $\Pi \equiv_u \Pi - \{r\} \cup r^{\rightarrow}$ if G-LSH(Π, r) .

Proof. $\Pi \equiv_u \Pi - \{r\} \cup r^{\rightarrow}$ iff for all finite sets of facts F: $\mathrm{Gr}(\Pi \cup F) \equiv \mathrm{Gr}(\Pi - \{r\} \cup r^{\rightarrow} \cup F)$. Define $\Delta := \mathrm{Gr}(\Pi \cup F) - \mathrm{Gr}(\Pi - \{r\} \cup F)$. Let $r'_1 \in \Delta$, i.e., there exists $\vartheta_1 : V_r \rightarrow \mathcal{D}_{\Pi \cup F}$ such that $r'_1 = r\vartheta_1$. If $\mathrm{Gr}(\Pi, \mathcal{D}_\Pi \cup \mathcal{D}_F)$ is head-cycle free then also $\mathrm{Gr}(\Pi \cup F)$ since F is a set of facts. From this and G-LSH(Π, r) it follows that LSH$(\mu(\mathrm{Gr}(\Pi \cup F)), \mu(r'_1))$ holds. Therefore, $\mathrm{Gr}(\Pi \cup F) \equiv \mathrm{Gr}(\Pi \cup F) - \{r'_1\} \cup (r'_1)^{\rightarrow}$ because of Proposition 10. Let $r'_2 \in \Delta$ and $r'_1 \neq r'_2$ if such an r'_2 exists. Since $(r'_1)^{\rightarrow}$ does not introduce any new positive dependencies it follows that $\mathrm{Gr}(\Pi \cup F) \equiv \mathrm{Gr}(\Pi \cup F) - \{r'_1, r'_2\} \cup (r'_1)^{\rightarrow} \cup (r'_2)^{\rightarrow}$ by the same argument as before. Repeating this argument yields $\mathrm{Gr}(\Pi \cup F) \equiv \mathrm{Gr}(\Pi \cup F) - \Delta \cup \Delta^{\rightarrow} = \mathrm{Gr}(\Pi - \{r\} \cup F) \cup \Delta^{\rightarrow} =: G$ where $\Delta^{\rightarrow} = (r'_1)^{\rightarrow} \cup ... \cup (r'_n)^{\rightarrow}$. Define $\Gamma := \mathrm{Gr}(\Pi - \{r\} \cup r^{\rightarrow} \cup F) - G$. Every $r' \in \Gamma$ is an instantiation of $r^{\rightarrow}$ and can be added to G because of Proposition 11. Therefore $\mathrm{Gr}(\Pi \cup F) \equiv \mathrm{Gr}(\Pi - \{r\} \cup r^{\rightarrow} \cup F)$. Note, that the active domain is not changed by the transformation. □

4.2 Complexity Analysis

We analyze the complexity of deciding G-LSH. Therefore, we first study the problem to decide if a program is head-cycle free.

Definition 18. Let Π be a program and $r \in \Pi$. $\mathrm{HCF}(\Pi, r)$ iff for all $C \subseteq_{fin} \mathcal{D}$ and $r' \in \mathrm{Gr}(r, C)$ it holds that r' is head-cycle free in $\mathrm{Gr}(\Pi, C)$. □

The following lemma shows that HCF is decidable by restricting C to be a subset of a finite set. We say that $\mathrm{Gr}(\Pi, C)$ contains an *r-head-cycle*, $r \in \mathrm{Gr}(\Pi, C)$, iff there exist distinct $a, b \in H(r)$, and a cycle $C(a, b)$ in $D^+(\mathrm{Gr}(\Pi, C))$ which starts and ends with a and contains b. Hence, r is head-cycle free in $D^+(\mathrm{Gr}(\Pi))$ iff $D^+(\mathrm{Gr}(\Pi))$ contains no r-head-cycle.

Lemma 2. Let Π be a program, $n := \max_{ar}(\Pi)$, and A, B be sets of constants such that $|A| = 2n$, $|B| = n$, and $A, B, \mathcal{D}_\Pi$ be pairwise disjoint. There exist $\mathcal{D}_\Pi \subseteq C \subseteq \mathcal{D}$, $r \in \mathrm{Gr}(\Pi, C)$, and an r-head-cycle in $\mathrm{Gr}(\Pi, C)$ iff there exists $r' \in \mathrm{Gr}(\Pi, \mathcal{D}_0)$, and an r'-head-cycle in $\mathrm{Gr}(\Pi, \mathcal{D}_0)$ with $\mathcal{D}_0 := \mathcal{D}_\Pi \cup A \cup B$.

Proof. $\Rightarrow$) In the following it is useful to assume that $C \cap (A \cup B) = \{\}$. If $C \cap (A \cup B) \neq \{\}$ replace every constant from $C \cap (A \cup B)$ in $A \cup B$ by a new and different constants such that A and B fullfill the prerequisites of the Lemma.

Define $D := D^+(\mathrm{Gr}(\Pi, C))$ and $D_0 := D^+(\mathrm{Gr}(\Pi, \mathcal{D}_0))$. Let $C(a, b)$ be an r-head-cycle in D. Denote by l the length of $C(a, b)$. We write the i-th edge in $C(a, b)$ as $(x_i\vartheta_i, y_i\vartheta_i)_{r_i}$: $x_i \in H(r_i)$, $y_i \in B^+(r_i)$, $\vartheta_i : V_{r_i} \to C$ a ground substitution, and therefore $(x_i\vartheta_i, y_i\vartheta_i) \in E(D)$. Here, $E(D)$ is the edge relation of the graph D. We will construct $\vartheta'_i : V_{r_i} \to \mathcal{D}_0$ for every $1 \leq i \leq l$ such that $C'(a', b') := ((x_1\vartheta'_1, y_1\vartheta'_1), (x_2\vartheta'_2, y_2\vartheta'_2), ..., (x_l\vartheta'_l, y_l\vartheta'_l))$ is a head-cycle in D_0.

Following the prerequisite there exists $x, y \in H(r)$, and $\vartheta : V_r \to C$ such that $a = x\vartheta$ and $b = y\vartheta$. Let $\Delta := (\mathcal{D}_{\{a\}} \cap \mathcal{D}_{\{b\}}) - \mathcal{D}_\Pi$ and $g : \Delta \to B$ be an injection; g exists since $|\Delta| \leq n$. Consider the first edge $(x_1\vartheta_1, y_1\vartheta_1)_{r_1}$ in $C(a, b)$. Let f_1 be an injection which maps to every constant in $\mathcal{D}_{\{x_1\vartheta_1, y_1\vartheta_1\}} - \mathcal{D}_\Pi - \Delta$ a constant from A. Such an injection exists since $|\mathcal{D}_{\{x_1\vartheta_1, y_1\vartheta_1\}}| \leq |A|$. Define $\vartheta'_1 := (f_1 \cup g) \circ \vartheta_1$. ϑ'_1 is of the form

$V_{r_1} \to \mathcal{D}_0$ and therefore $(x_1\vartheta_1', y_1\vartheta_1') \in E(D_0)$.

There exists a second edge in $C(a,b)$ since $a \neq b$. We have to find $\vartheta_2' : V_{r_2} \to \mathcal{D}_0$ such that $y_1\vartheta_1' = x_2\vartheta_2'$. Let f' be f_1 with $\mathrm{dom}(f_1)$ restricted to constants from $\mathcal{D}_{\{x_2\vartheta_2\}} - \mathcal{D}_\Pi - \Delta$, i.e., $f'(c) = f_1(c)$ if $c \in \mathcal{D}_{\{x_2\vartheta_2\}} - \mathcal{D}_\Pi - \Delta$. Extend f' to $f_2 : (\mathcal{D}_{\{x_2\vartheta_2, y_2\vartheta_2\}} - \mathcal{D}_\Pi - \Delta) \to A$ such that f_2 is injective. There are enough constants in A for this. Define $\vartheta_2' := (f_2 \cup g) \circ \vartheta_2$; ϑ_2' is of the form $V_{r_2} \to \mathcal{D}_0$. Again $(x_2\vartheta_2', y_2\vartheta_2') \in E(D_0)$. Note, that $y_1\vartheta_2' = x_2\vartheta_2'$. Now we can repeat this step and construct the head-cycle $C'(a',b')$: In the i-th step f' is the restriction of f_{i-1} to constants from $\mathcal{D}_{\{x_i\vartheta_i\}} - \mathcal{D}_\Pi - \Delta$, and $f_i : (\mathcal{D}_{\{x_i\vartheta_i, y_i\vartheta_i\}} - \mathcal{D}_\Pi - \Delta) \to A$ is an injective extension of f'.

At last we show that there exists a ground substitution $\vartheta' : V_r \to \mathcal{D}_0$ such that $a' = x\vartheta'$ and $b' = y\vartheta'$. Let r' be $r\vartheta'$. Since $x, y \in H(r)$ it follows that $a', b' \in H(r')$, and therefore, $C'(a',b')$ is a head-cycle in D_0. Recall that Δ contains the constants which are both in a and b and not in Π. Throughout the construction these constants are mapped via g to constants from B. Extend g to $f : V_r \to \mathcal{D}_0$ such that $a' = x(f \circ \vartheta)$ and $b' = y(f \circ \vartheta)$. Define $\vartheta' := (f \circ \vartheta)$.

$\Leftarrow$) Clear. □

With this result at hand, we now show that G-LSH can be decided in PSPACE. We start to show this result for the condition HCF, and briefly argue latter on that the remaining condition from Definition 17 is not harder to decide.

Theorem 8. Deciding

$$\{(\Pi, r) : \mathrm{HCF}(\Pi, r) \text{ holds, } \Pi \text{ a program, } r \in \Pi\}$$

is complete for PSPACE.

Proof. It holds that, deciding $\mathrm{HCF}(\Pi, r)$ is complete for PSPACE iff deciding $\overline{\mathrm{HCF}}(\Pi, r)$ is complete for PSPACE. We are considering $\overline{\mathrm{HCF}}(\Pi, r)$, i.e., if there exists $r' \in \mathrm{Gr}(\Pi, \mathcal{D}_0)$ and an r'-head-cycle in $\mathrm{Gr}(\Pi, \mathcal{D}_0)$ (cf. Lemma 2).

Membership. Define $P := \{(\Pi, r) : \overline{\mathrm{HCF}}(\Pi, r)\}$. The following algorithm shows that $P \in \mathrm{NPSPACE}$, which implies $P \in \mathrm{PSPACE}$.

1. Let n be the size of Π and $k := \max_{ar}(\Pi)$.
2. For all $a, b \in H(r)$: {
3. Nondeterministically instantiate a, b with constants from $\mathcal{D}_0$ to a', b'.
4. Reject if $a' = b'$.
5. Set h to 0.
6. Set z to a'.
7. For $i = 1, ..., (4 \cdot n)^{k+1}$: {
8. Nondeterministically choose $s \in \Pi$.
9. For all $x \in H(s)$, $y \in B^+(s)$: {
10. Nondet. instantiate x, y with constants from $\mathcal{D}_0$ to x', y'.
11. If $z \neq x'$ then reject.
12. Set z to y'.
13. If $z = a'$ and $h = 1$ then accept.
14. If $z = b'$ then set h to 1.
15. }
16. }}
17. Reject.

Claim. The algorithm accepts (Π, r) iff $(\Pi, r) \in P$.

Denote by $z_1, z_2, ..., z_l$ the values of z in the computation of M on input (Π, r). Assume, the algorithm accepts (Π, r). This is the case if $z_1 = z_l = a'$ and $z_j = b'$ for some j. The latter is indicated by the value of h. Therefore, $z_1, z_2, ..., z_l$ induces an r'-head-cycle in $D_0 := D^+(\mathrm{Gr}(\Pi, \mathcal{D}_0))$ for some instantiation r' of r, i.e., $(\Pi, r) \in P$. Now assume, $(\Pi, r) \in P$. From Lemma 2 it follows that $(\Pi, r) \in P$ iff there exists an r'-head-cycle in D_0. The algorithm nondeterministically searches all paths in D_0 of length at most $(4 \cdot n)^{k+1}$. The number of different vertices in D_0 is at most $(4 \cdot n)^{k+1}$, because there are at most n different relational symbols in Π, and the number of different instantiations of an atom is bounded by $(4 \cdot n)^k$. For the latter, recall that $|\mathcal{D}_0| = |\mathcal{D}_\Pi| + 3 \cdot k \leq 4 \cdot n$. This shows $|V(D_0)| \leq (4 \cdot n)^{k+1}$, and that the algorithm accepts (Π, r).

Claim. Let n and k be as in the algorithm, and s_M be the space complexity of M. Then $s_M(n) = O(\lg(n) \cdot k)$.

The loops over $H(r)$, $H(s)$, $B^+(s)$, and the choice of $s \in \Pi$ can be implemented by pointers to the input with size $O(\lg(n))$. The elements of $\mathcal{D}_0$ can be represented with size $O(\lg(n))$ since $|\mathcal{D}_0| = |\mathcal{D}_\Pi| + 3 \cdot k \leq 4 \cdot n = O(n)$. For every relational symbol a in Π: $\max_{ar}(a) \leq k$. Therefore, the instantiation of a, b, x, and y needs size $O(\lg(n) \cdot k)$. The loop variable i needs size $\lg((4 \cdot n)^{k+1}) = O(\lg(n) \cdot k)$.

Hardness. Let $M = \langle Q, \Sigma, q_0, q_a, q_r, \delta \rangle$ be a polynomial space bounded Turing machine where Q is a finite set of states, Σ a finite set of symbols, $\rhd, \sqcup \notin Q \cup \Sigma$ and $Q \cap \Sigma = \{\}$, $q_0 \in Q$ is the initial state, $q_a \in Q$ is the accepting state, $q_r \in Q$ is the rejecting state and $\delta : Q \times \Sigma \to \Sigma \times \{\leftarrow, |, \rightarrow\} \times Q$ is the transition function. The Turing machine has one tape with the left delimiter $\rhd$ and the blank symbol $\sqcup$. The space complexity $s_M(n)$ of M is bounded by a polynomial p over $\mathbb{N}$, i.e., for all $n \in \mathbb{N} : s_M(n) \leq p(n)$.

Define for the input $x \in \Sigma^*$ the initial configuration as follows: M is in q_0, the tape contains $\rhd x \sqcup \sqcup \sqcup ...$ and the r/w head points to $\rhd$. Final configurations: M is either in q_a or q_r, the tape contains $\rhd \sqcup \sqcup \sqcup ...$ and the r/w head points to $\rhd$. A configuration for M can be represented by a string $s \in (\Sigma \cup Q \cup \{\rhd, \sqcup\})^*$ with $|s| = 2 + p(n)$. We have to consider the left delimiter $\rhd$ and one symbol which represents the current state and the position of the r/w head. This symbol is on the left to the symbol the r/w head points to.

The reduction f maps to (M, x) a program Π and a rule r such that M accepts x iff $\overline{\mathrm{HCF}}(\Pi, r)$. Let u be a $2+p(|x|)$-place relational symbol. Define $u_0 := u(q_0, \rhd, x_1, ..., x_{|x|}, \sqcup, \sqcup, \sqcup, ...)$ and $u_a := u(q_a, \rhd, \sqcup, \sqcup, \sqcup, ...)$. The rule r is the disjunctive fact $u_0 \vee u_a \leftarrow$. The rules $u_0 \vee u_a \leftarrow$ and $u_a \leftarrow u_0$ are in Π. If $\delta(q, \sigma) = (\sigma', \leftarrow, q')$ then for every $1 < i < 2 + p(|x|)$ the rule

$$u(..., X_{i-1}, q, \sigma, ...) \leftarrow u(..., q', X_{i-1}, \sigma', ...)$$

is in Π. If $\delta(q, \sigma) = (\sigma', |, q')$ then for every $1 < i < 2 + p(|x|)$ the rule

$$u(..., X_{i-1}, q, \sigma, ...) \leftarrow u(..., X_{i-1}, q', \sigma', ...)$$

is in Π. If $\delta(q, \sigma) = (\sigma', \rightarrow, q')$ then for every $1 < i < 2 + p(|x|)$ the rule

$$u(..., X_{i-1}, q, \sigma, ...) \leftarrow u(..., X_{i-1}, \sigma', q', ...)$$

is in Π. The rules for $i = 1$ and $i = 2 + p(n)$ if admissible are special case which we will not consider here. Note, that these rules are safe. The size

of Π is polynomial in the size of (M, x) and f is computable in logarithmic space.

Consider the dependency graph $D = D^+(\mathrm{Gr}(\Pi, C))$ for some $C \supseteq \mathcal{D}_\Pi$. There exists exactly one vertex w with $(u_0, w) \in E(D)$ since there exists exactly one configuration after the initial configuration and because of the construction of (Π, r). The same holds for every configuration in the computation of M on x. For seeing this just note, that the string representation of a configuration k has exactly one $q \in Q$ in it and that every atom in Π has exactly one symbol $q \in Q$ in it. So, there exists a *unique* path from u_0 to either u_a or $u(q_r, \rhd, \sqcup, \sqcup, \sqcup, ...)$.

Assume that M accepts x. There exists a path from u_0 to u_a and an edge (u_a, u_0) in $D^+(\mathrm{Gr}(\Pi))$. Therefore, $D^+(\mathrm{Gr}(\Pi))$ contains a head-cycle $C(u_0, u_a)$, which implies $\overline{\mathrm{HCF}}(\Pi, r)$. Now, assume $\overline{\mathrm{HCF}}(\Pi, r)$. Since $\mathrm{Gr}(r, C) = \{u_0 \vee u_a \leftarrow\}$ for any C, and r is the only rule in Π with $|H(r)| > 1$, there are a two possible cases: $\exists C(u_0, u_a)$ or $\exists C(u_a, u_0)$. Both cases imply that there exists a path $P(u_0, u_a)$ from u_0 to u_a. The path $P(u_0, u_a)$ induces an accepting computation of M for x. □

The previous result also holds for G-LSH. The upper bound holds since deciding if $|H(r')| > 1$ holds for all r', r' as in the definition of G-LSH, is an unification problem which is linear time decidable. Another way to see this is to "accept" in line 4 of the algorithm of the previous proof. The lower bound holds since HCF is reducible to G-LSH.

Theorem 9. Deciding

$$\{(\Pi, r) : \text{G-LSH}(\Pi, r) \text{ holds, } \Pi \text{ a program, } r \in \Pi\}$$

is complete for PSPACE.

Example 15. Let Π be

$$b(X) \leftarrow a(X)$$
$$a(X) \vee b(X) \leftarrow c(X)$$

For every $\mathcal{D}_\Pi \subseteq C \subseteq \mathcal{D}$ the program $\mathrm{Gr}(\Pi, C)$ is head-cycle free. Therefore

Π is uniformly equivalent to

$$b(X) \leftarrow a(X)$$
$$a(X) \leftarrow c(X), \text{not } b(X)$$
$$b(X) \leftarrow c(X), \text{not } a(X)$$

because also $a(c) \neq b(c)$ for every $c \in \mathcal{D}$. □

Now we consider the problems if we bound the maximal arity of the programs by a constant. Define $\text{HCF}_k(\Pi, r)$ as $\text{HCF}(\Pi, r) + (\max_{ar}(\Pi) \leq k)$.

Theorem 10. Let $k \in \mathbb{N}$. Deciding

$$\{(\Pi, r) : \text{HCF}_k(\Pi, r), \Pi \text{ a program}, r \in \Pi\}$$

is complete for NL.

Proof. We will show that $\overline{\text{HCF}}_k$ is complete for coNL = NL.

Membership. Consider the algorithm from the previous proof. It runs in space $O(\lg(n) \cdot k) = O(\lg(n))$ since k is constant here.

Hardness. The reachability problem in digraphs can be reduced to $\overline{\text{HCF}}_0$ which is a special case of $\overline{\text{HCF}}_k$. □

Define $\text{G-LSH}_k(\Pi, r)$ as $\text{G-LSH}(\Pi, r) + (\max_{ar}(\Pi) \leq k)$. The correctness of this rule is a consequence of Theorem 7. Again, the previous theorem also holds for $\text{G-LSH}_k(\Pi, r)$.

Theorem 11. Let $k \in \mathbb{N}$. Deciding

$$\{(\Pi, r) : \text{G-LSH}_k(\Pi, r), \Pi \text{ a program}, r \in \Pi\}$$

is complete for NL.

Proof. Membership. We already have shown that HCF_k is in NL. Deciding the set

$$\{r : \exists r' \text{ an instantiation of } r \text{ such that } |H(r')| \leq 1\}$$

can be done in nondeterministic space by iterating with a pointer to the input through the head and writing down nondeterministically the values of the instantiated variables. Since coNL = NL also the complementary problem is in NL and hence, G-LSH$_k$ is in NL.

Hardness. The reachability problem in digraphs can be reduced to $\overline{\text{G-LSH}_0}$. Since $k = 0$ here, it holds that there exists an instantiation r' of $r \in \Pi$ for a program Π with $|H(r')| \leq 1$ iff there are two atoms in $H(r)$ with different relational symbols, i.e., $|H(r)| \leq 1$. □

In many applications it seems that the arity does not exceed, say, 5. This number is arbitrary here. But it rarely occurs that the arity grows linear with the input size. Therefore, Theorem 11 says that in many applications G-LSH(Π, r) is efficiently decidable.

Chapter 5

Implementations

In this chapter we provide implementations for

- deciding strong equivalence of programs,
- deciding SUBS*, and
- deciding G-LSH.

The algorithms are essentially reductions, i.e., the reduce the problem to some already implemented problem. Testing strong equivalence is reduced to the unsatisfiability problem of Bernays-Schönfinkel formulas. Deciding SUBS* is reduced to the Boolean conjunctive query problem. And deciding G-LSH is reduced to the brave reasoning problem of disjunctive datalog programs with negation. The first algorithm is written in C++ and additionally uses the automated theorem prover Darwin[1]. The other algorithms are written in Perl and use the system DLV[2] for evaluating disjunctive datalog programs with negation. The algorithm for testing strong equivalence is presented in Section 5.1 and the other algorithms are presented in Section 5.2.

[1] http://goedel.cs.uiowa.edu/Darwin/
[2] http://www.dlvsystem.com

5.1 Strong Equivalence Test

Lin [Lin02] showed how to reduce the problem of testing strong equivalence to the unsatisfiability problem of Bernays-Schönfinkel formulas. In the next paragraph we introduce this problem and refine the reduction. In the last paragraph we discuss implementation issues. Roughly spoken, two DLV programs Π_1 and Π_2 are translated into two Darwin clausal forms φ_1 and φ_2 such that Π_1 and Π_2 are strongly equivalent iff φ_1 and φ_2 are unsatisfiable. Darwin is an automated theorem prover. Baumgartner and Tinelli [BT03] pointed out that the model evolution calculus terminates for Bernays-Schönfinkel formulas. Darwin implements this calculus and therefore halts on Bernays-Schönfinkel formulas unlike most other automated theorem provers. We also remark here that we need the refinement because the reduction of Lin [Lin02] does not use clausal form which is needed for most automated theorem provers.

Algorithm for Testing Strong Equivalence First we define Bernays-Schönfinkel formulas. For a quantifier free formula φ without function and constant symbols a *Bernays-Schönfinkel formula* is of the form

$$\exists x_1 ... x_k \forall y_1 ... y_l \varphi(x_1, ..., x_k, y_1, ..., y_l)$$

For such a formula ψ the satisfiability problem is complete for NEXPTIME. The *clausal form* of ψ is obtained from ψ by replacing the variables $x_1, ..., x_k$ in φ with new constant symbols, transforming the resulting formula φ' into a conjunctive normal form φ'', and putting all clauses of φ'' in a set $\Sigma(\bar{x})$ where $\bar{x}$ are the free variables of φ''. The clausal form $\Sigma(\bar{x})$ is satisfiable iff ψ is satisfiable. We say that a clausal form $\Sigma(\bar{x})$ is *satisfiable* iff $\forall \bar{x} \Sigma(\bar{x})$ is satisfiable. The notion

$$l_1, ..., l_k \vdash l_{k+1}, ..., l_n$$

stands for the clause

$$l_1 \vee ... \vee l_k \vee \neg l_{k+1} \vee ... \vee \neg l_n.$$

Next we describe a reduction of the complementary problem of deciding strong equivalence to the Bernays-Schönfinkel satisfiability problem. This resembles the reduction by Lin [Lin02].

For every relational symbol r_i, $1 \leq i \leq k$, of the programs Π_1, Π_2 let r_i' be a new relational symbol of the same arity. Then

$$\Sigma(\bar{x}_1, ..., \bar{x}_k) := \{r_1'(\bar{x}_1) \vdash r_1(\bar{x}_1), ..., r_k'(\bar{x}_k) \vdash r_k(\bar{x}_k)\}.$$

Here we assume that $\bar{x}_i$ and $\bar{x}_j$ are disjoint for $i \neq j$ without loss of generality since we can rename the variables of a rule arbitrary as long as different variables remain different. For each rule

$$h_1(u_1) \vee ... \vee h_l(u_l) \leftarrow p_1(v_1), ..., p_m(v_m), \text{not } p_{m+1}(v_{m+1}), ..., \text{not } p_n(v_n)$$

in Π, let $\Gamma_\Pi(\bar{u}, \bar{v})$ contain

$$h_1(u_1), ..., h_l(u_l) \vdash p_1(v_1), ..., p_m(v_m), \neg p_{m+1}'(v_{m+1}), ..., \neg p_n'(v_n)$$

and

$$h_1'(u_1), ..., h_l'(u_l) \vdash p_1'(v_1), ..., p_m'(v_m), \neg p_{m+1}'(v_{m+1}), ..., \neg p_n'(v_n).$$

Let U be the set of unique names axioms, i.e. U asserts that no two constants are equal. Therefore a new relational symbol U_i for every constant symbol c_i is introduced:

$$\begin{aligned} &U_1(c_1) \wedge \neg U_2(c_1) \wedge ... \wedge \neg U_k(c_k) \\ &\neg U_1(c_1) \wedge U_2(c_1) \wedge ... \wedge \neg U_k(c_k) \\ &\vdots \\ &\neg U_1(c_l) \wedge \neg U_2(c_1) \wedge ... \wedge U_k(c_k) \end{aligned}$$

For a formula φ, let φ_y^x be the formula with y replaced by x. Analogous for a set of formulas. Let $\bar{c}$ be $c_1, ..., c_k$. Lin [Lin02] showed that Π and Π' are strongly equivalent iff

$$\forall\bar{u}\forall\bar{y}\exists\bar{x}\exists\bar{z}(\neg U_{\bar{c}}^{\bar{u}} \vee \neg\Sigma(\bar{z}) \vee \neg\Gamma_\Pi(\bar{x})_{\bar{c}}^{\bar{u}} \vee \Gamma_{\Pi'}(\bar{y})_{\bar{c}}^{\bar{u}})$$

and

$$\forall\bar{u}\forall\bar{y}\exists\bar{x}\exists\bar{z}(\neg U_{\bar{c}}^{\bar{u}} \vee \neg\Sigma(\bar{z}) \vee \neg\Gamma_{\Pi'}(\bar{x})_{\bar{c}}^{\bar{u}} \vee \Gamma_\Pi(\bar{y})_{\bar{c}}^{\bar{u}})$$

are valid. Therefore, Π and Π' are *not* strongly equivalent iff

$$\exists\bar{u}\exists\bar{y}\forall\bar{x}\forall\bar{z}(U_{\bar{c}}^{\bar{u}} \wedge \Sigma(\bar{z}) \wedge \Gamma_\Pi(\bar{x})_{\bar{c}}^{\bar{u}} \wedge \neg\Gamma_{\Pi'}(\bar{y})_{\bar{c}}^{\bar{u}})$$

or

$$\exists\bar{u}\exists\bar{y}\forall\bar{x}\forall\bar{z}(U_{\bar{c}}^{\bar{u}} \wedge \Sigma(\bar{z}) \wedge \Gamma_{\Pi'}(\bar{x})_{\bar{c}}^{\bar{u}} \wedge \neg\Gamma_\Pi(\bar{y})_{\bar{c}}^{\bar{u}})$$

is satisfiable.

The sets U, Σ, $\Gamma_\Pi(\bar{x})$, $\Gamma_{\Pi'}(\bar{y})$ are in clausal form, but $\neg\Gamma_\Pi(\bar{x})$, $\neg\Gamma_{\Pi'}(\bar{y})$ are not. They are in quantifier free disjunctive normal form. It is sufficient to use satisfiability equivalent formulas to gain clausal form. Denote these formulas by $\Gamma^*_\Pi(\bar{x})$, $\Gamma^*_{\Pi'}(\bar{y})$. It follows that Π and Π' are not strongly equivalent iff

$$\exists\bar{u}\exists\bar{y}\forall\bar{x}\forall\bar{z}(U^{\bar{u}}_{\bar{c}} \wedge \Sigma(\bar{z}) \wedge \Gamma_\Pi(\bar{x})^{\bar{u}}_{\bar{c}} \wedge \Gamma^*_{\Pi'}(\bar{y})^{\bar{u}}_{\bar{c}})$$

or

$$\exists\bar{u}\exists\bar{y}\forall\bar{x}\forall\bar{z}(U^{\bar{u}}_{\bar{c}} \wedge \Sigma(\bar{z}) \wedge \Gamma_{\Pi'}(\bar{x})^{\bar{u}}_{\bar{c}} \wedge \Gamma^*_{\Pi}(\bar{y})^{\bar{u}}_{\bar{c}})$$

is satisfiable. By Skolemization, this is the case iff

$$U \cup \Sigma(\bar{z}) \cup \Gamma_\Pi(\bar{x}) \cup \Gamma^*_{\Pi'}$$

or

$$U \cup \Sigma(\bar{z}) \cup \Gamma_{\Pi'}(\bar{x}) \cup \Gamma^*_{\Pi}$$

is satisfiable. In the following denote $U \cup \Sigma(\bar{z}) \cup \Gamma_\Pi(\bar{x}) \cup \Gamma^*_{\Pi'}$ by $\Delta(\Pi, \Pi')$.

Now we show by an example how to get Γ^* from a quantifier free disjunctive normal form Γ. Let Γ be $A \vee (B \wedge C)$ and s be a new unary relational symbol. Then Γ^* is $(A \vee s(c)) \wedge (B \vee \neg s(c)) \wedge (C \vee \neg s(c))$ where c does not occur in Γ. Therefore $\forall\bar{x}(\Gamma(\bar{x}))$ is satisfiable iff $\forall\bar{x}(\Gamma^*(\bar{x}))$ is satisfiable. The algorithm for computing Γ^* from Γ in general is defined as follows. It assumes that the variables in different terms of the DNF, i.e. the variables in the conjunctions in the DNF, are disjoint. This holds for the DNF's we consider.

Algorithm: A0
Input: a quantifier free DNF Γ, such that the variables in different terms are disjoint
Output: a clausal form Γ^*

1. $\Gamma^* := \{\}$, $\varphi := \bot$, and let s be a new unary relational symbol.
2. For every $t \in \Gamma$:
3. Let t be $l_1 \wedge ... \wedge l_k$, and c be a new constant symbol.
4. Add $l_1 \vee \neg s(c), ..., l_k \vee \neg s(c)$ to Γ^*.
5. Set φ to $\varphi \vee s(c)$.
6. Delete t from Γ.

7. Add φ to Γ^*. □

Proposition 12. (Correctness) Let Γ be a a quantifier free DNF and Γ^* be the output of algorithm A0 on input Γ. Then Γ^* is satisfiable iff Γ is satisfiable.

Proof. By construction, Γ^* is in quantifier free CNF. Assume that Γ^* is satisfiable. Since $\varphi \in \Gamma^*$, it holds that $s(c)$ is true for some c. It follows that the $l_1, ..., l_k$ which were added in the step when c was used are true, i.e. $l_1 \wedge ... \wedge l_k$ is true, and therefore Γ is satisfiable. On the other hand, let M be a model of Γ. There exists a true term in Γ for which a constant symbol c in the construction of Γ^* was used. Let M^* be the structure obtained from M such that additionally $s(c)$ holds in M^*. Since the used constants do not occur in Γ it holds that M^* is a model of Γ^*. □

Note that no function symbols of positive arity were introduced.

The following algorithm reduces the strong equivalence problem to the unsatisfiability problem of Bernays-Schönfinkel formulas. Its correctness follows from what we said previously.

Algorithm: A1
Input: programs P and Q
Output: two clausal forms C_1, C_2

1. Compute $C_1 := \Delta(P, Q)$.
2. Compute $C_2 := \Delta(Q, P)$.
3. Output (C_1, C_2). □

Proposition 13. (Correctness) Let P and Q be programs and C_1, C_2 be the clausal forms computed by A1 on input (P, Q). Then $P \equiv_s Q$ iff C_1 and C_2 are unsatisfiable.

The algorithm obviously works in polynomial time. For estimating the size of the output we need a more detailed output language which is introduced in the next paragraph.

Implementation Issues We first describe the syntax of DLV and Darwin and later give an estimation of the output size of algorithm A1.

The used DLV syntax in EBNF:

```
rule ::= head '.'  | head ':-' body '.'  | ':-' body '.'
head ::= literal {'v' literal}
body ::= literal {',' literal}
literal ::= atom | 'not' atom
atom ::= symb ['(' term {',' term}')']
term ::= variable | symb | num
variable ::= ('A'-'Z' ) {'a'-'z' | 'A'-'Z' | '0'-'9'}
symb ::= ('a'-'z') {'a'-'z' | 'A'-'Z' | '0'-'9'}
num ::= ('0'-'9') {'0'-'9'}
```

Let L_{DLV} be the language given by this syntax description.

Remarks.

- Equality is not supported. There is no = symbol in L_{DLV} .
- Comments are not supported. There is no % symbol.
- Anonymous variables are not supported. There is no _ in L_{DLV} .
- The set of atoms in L_{DLV} is a proper subset of the set of atoms in L_{Darwin}.

The used Darwin syntax in EBNF:

```
clause ::= head [':-' body].
head ::= literal {';' literal}
body ::= literal {',' literal}
literal ::= [('-' | '∼')] atom | 'true' | 'false'
atom ::= symb | symb '(' term {',' term} ')'
           | '(' term '=' term ')'
term ::= variable | symb ['(' term {',' term} ')']
variable ::= ('A'-'Z' | '_') {'a'-'z' | 'A'-'Z' | '0'-'9' | '_'}
symb ::= ('a'-'z' | '0'-'9') {'a'-'z' | 'A'-'Z' | '0'-'9' | '_'}
```

Let L_{Darwin} be the language given by this syntax description. With these languages we can represent the sets Π, Π', U, Σ, ... Their size $|\cdot|$ is the number of symbols used.

The size of Γ_Π is linear in $|\Pi|$. The size of Γ^*_Π is linear in the size of Γ_Π. Let n_c be the number of constant symbols, and n_r be the number of relational symbols in Π and Π'. The size of Σ is linear in n_r and the size of U is quadratic in n_c. The following proposition follows directly from the construction of $\Delta(\Pi, \Pi')$.

Proposition 14. $|\Delta(\Pi, \Pi')| \leq e \cdot (|\Pi| + |\Pi'| + n_r + n_c^2)$ for some $e \in \mathbb{N}$.

Algorithm A1 is implemented in the programming language C++ and uses the automated theorem prover Darwin. Although there are other theorem provers with a similar input language as L_{Darwin} one has to be careful because it is not guaranteed in general that a theorem prover halts on an Bernays-Schönfinkel input. Darwin does this. It is also clear that the overall performance of testing strong equivalence depends heavily on the automated theorem prover since algorithm A1 is rather efficient and the complex problem to solve is checking unsatisfiability of Bernays-Schönfinkel formulas.

We end this paragraph with an example of how the implementation works.

Example 16. Let Π (in DLV syntax) be

```
a(k1).
a(k2).
 h(X):- a(X).
 t(X):- h(X).
 a(X):- t(X).
 a(X):- h(X).
```

The program Π states that $a \subseteq h \subseteq t \subseteq a$, i.e. $a = h = t$. Therefore the last rule is useless.

Let Π' be

```
a(k1).
a(k2).
 h(X):- a(X).
 t(X):- h(X).
 a(X):- t(X).
```

In the following the single parts of the resulting formula $\Delta(\Pi, \Pi')$ are given in Darwin syntax.

Σ:
```
 a_(X1):-a(X1).
 t_(X1):-t(X1).
 h_(X1):-h(X1).
```

$\Gamma(\Pi)$:
```
 a(k1).  a(k2).  a_(k1).  a_(k2).
 h(X):- a(X).    h_(X):- a_(X).
 t(X):- h(X).    t_(X):- h_(X).
 a(X):- t(X).    a_(X):- t_(X).
 a(X):- h(X).    a_(X):- h_(X).
```

U:
```
 u1(k1).  -u1(k2).
 -u2(k1).  u2(k2).
```

Using Darwin a refutation is found. This also holds for $\Delta(\Pi', \Pi)$. Hence Π' and Π are strongly equivalent. □

5.2 Simplifier

In this section we will show how to handle SUBS$^*(r, s)$ and G-LSH(Π, r) in practice. The problem is to decide if the rule is applicable. We will reduce both problems to evaluation problems of disjunctive datalog programs with

$\Gamma^*(\Pi')$:

```
-a(k1):- s__(1).        -a_(k1):- s__(6).
-a(k2):- s__(2).        -a_(k2):- s__(7).
-h(sk_1):- s__(3).      -h_(sk_4):- s__(8).
a(sk_1):- s__(3).       a_(sk_4):- s__(8).
-t(sk_2):- s__(4).      -t_(sk_5):- s__(9).
h(sk_2):- s__(4).       h_(sk_5):- s__(9).
-a(sk_3):- s__(5).      -a_(sk_6):- s__(0).
t(sk_3):- s__(5).       t_(sk_6):- s__(0).

s__(1), s__(2), s__(3),
s__(4), s__(5), s__(6),
s__(7), s__(8), s__(9),
s__(0).
```

negation which can be solved by the DLV[3] system. The next two paragraphs are dedicated to algorithms for deciding SUBS$^*(r, s)$ and G-LSH(Π, r) respectively. In the last paragraph we discuss implementation issues, e.g., we shortly present the prototype of a toolbox we named Simplifier.

Algorithm for SUBS$^*(r, s)$ We will reduce SUBS$^*(r, s)$ to the Boolean conjunctive query problem which is defined as follows.

Definition 19. A rule q is a *Boolean conjunctive query* iff it is of the form

$$b \leftarrow a_1, ..., a_m$$

where the a_i's are atoms, and b is a propositional atom. □

The Boolean conjunctive query problem (BCQ) is the problem to decide if for a set F of facts with $b \notin F$ and a Boolean conjunctive query q there exists a stable model I of $F \cup \{q\}$ such that $b \in I$. Note that for all F and q the program $F \cup \{q\}$ has a stable model. This stable model is unique because $F \cup \{q\}$ is Horn and every Horn program has at most one stable model. It also coincides with the unique subset-minimal classical model of $F \cup \{q\}$. Hence, the BCQ problem is equivalent to the problem to decide if

[3] http://www.dlvsystem.com

there exists a variable assignment $\theta : V_q \to \mathcal{D}_F$ such that $B(q\theta) \subseteq F$ because this is the only case that b becomes true.

Proposition 15. $(q, F) \in$ BCQ iff $\exists\theta : V_q \to \mathcal{D}_F$ such that $B(q\theta) \subseteq F$.

The following algorithm reduces SUBS* to BCQ. We only have to take care of that

$$B^+(s\theta) \subseteq B^+(r), B^-(s\theta) \subseteq B^-(r), H(s\theta) \subseteq H(r) \cup B^-(r)$$

holds. For this purpose the algorithm introduces new relational symbols.

Algorithm: A2
Input: rules r, s
Output: Boolean conjunctive query q, set of facts F

1. $A_1 := H(s)$, $B_1 := H(r) \cup B^-(r)$,
   $A_2 := B^+(s)$, $B_2 := B^+(r)$,
   $A_3 := B^-(s)$, $B_3 := B^-(r)$.
2. Replace every relational symbol in $A_2 \cup B_2$ by a symbol which does not occur in $A_1 \cup B_1$ such that different symbols remain different.
3. Replace every relational symbol in $A_3 \cup B_3$ by a symbol which does not occur in $A_1 \cup B_1 \cup A_2 \cup B_2$ such that different symbols remain different.
4. Set $A := A_1 \cup A_2 \cup A_3$, $B := B_1 \cup B_2 \cup B_3$.
5. Replace every variable in B by a constant which does not occur in $A \cup B$ such that different symbols remain different.
6. Let b be a propositional atom which does not occur in $A \cup B$.
7. Set $q := (b \leftarrow A)$, $F := B$.
8. Output (q, F). □

Example 17. Let r be

$$e(X, Y) \vee e(Y, Z) \leftarrow e(Y, X), g(Z, X), \text{not } e(X, X)$$

and s be

$$e(W, V) \vee e(V, V) \leftarrow g(Z, V), \text{not } e(W, W).$$

For example, A might look like

$$\{e(W,V), e(V,V), g_p(Z,V), e_n(W,W)\}$$

and B like

$$\{e(1,2), e(2,3), e(1,1), e_p(2,1), g_p(3,1), e_n(1,1)\}.$$

The relational symbols e_n, e_p, and g_p are new symbols relative to r and s. The variable assignment $\theta := \{V \mapsto 1, W \mapsto 1, Z \mapsto 3\}$ makes the query $b \leftarrow A$ true with respect to B. The corresponding substitution ϑ is $\{V \mapsto X, W \mapsto X, Z \mapsto Z\}$. □

Theorem 12. SUBS* is polynomial time reducible to BCQ.

Proof. Let r and s be rules. Recall that SUBS$^*(r,s)$ holds iff there exists a substitution ϑ of the form $V_s \to V_r \cup \mathcal{D}_r$ such that

$$B^+(s\vartheta) \subseteq B^+(r), B^-(s\vartheta) \subseteq B^-(r), H(s\vartheta) \subseteq H(r) \cup B^-(r). \tag{5.1}$$

Algorithm A2 assigns to every pair (r,s) of rules a Boolean conjunctive query q and a set of facts F such that a variable assignment θ directly corresponds with a substitution ϑ: Let $\nu : V_r \to \mathcal{D}$ be the bijection created implicitly in step 5 of algorithm A2. Assume that $\exists\theta : V_q \to \mathcal{D}_F$ with $B(q\theta) \subseteq F$. Define $\vartheta := \nu^{-1} \circ \theta$. Then (5.1) holds because of the replacement of relational symbols and the construction of θ in A2. On the other side assume that such a ϑ which fulfills (5.1) exists. Then $\theta := \nu \circ \vartheta$ is a variable assignment with $B(q\theta) \subseteq F$. And from Proposition 15 we follow that there exists $\theta : V_q \to \mathcal{D}_F$ such that $B(q\theta) \subseteq F$ iff $(q,F) \in$ BCQ and hence, $(q,F) \in$ BCQ iff SUBS$^*(r,s)$ holds. □

The output size of algorithm A2 is linear in the input size. Implementation issues will be discussed in the last paragraph of this section.

Algorithm for G-LSH(Π, r) In this paragraph we present two algorithms for deciding G-LSH. The first calls several times an oracle of the brave reasoning problem and the second algorithm directly reduces G-LSH to the brave reasoning problem which will be introduced subsequently.

Assume that we have an algorithm A which decides if a ground atom t is reachable from a ground atom s in the positive dependency graph $D^+(\mathrm{Gr}(\Pi, \mathcal{D}_0))$ where $\mathcal{D}_0$ is as in Lemma 2 and $\mathcal{D}_s \subseteq \mathcal{D}_0$, $\mathcal{D}_t \subseteq \mathcal{D}_0$. The algorithm A takes as input Π, s, and t. Using A we can decide if r is head-cycle free in Π. For all distinct $s, t \in \mathrm{Gr}(\Pi, \mathcal{D}_0)$ we test if $A(\Pi, s, t)$ and $A(\Pi, t, s)$. If there exists such ground atoms then r is not head-cycle free in Π. Otherwise r is head-cycle free in Π.

We will need the following operators. Let U be a set of variables or constants and $p \in \mathcal{R}^1$. Define

$$\alpha(p, U) := \{p(x) : x \in U\}$$

Let r be a rule with $H(r) \neq \{\}$ and $B^+(r) \neq \{\}$. Define

$$\beta(r, p) := \{(b \leftarrow a, \alpha(p, V_b)) : a \in H(r), b \in B^+(r)\}$$

and for some $p \in \mathcal{R}^1 - \mathcal{R}_\Pi$

$$\gamma(\Pi, p) := \bigcup \{\beta(r, p) : r \in \Pi, H(r) \neq \{\}, B^+(r) \neq \{\}\}.$$

Example 18. Let Π be

$$\begin{aligned} a(1) & \\ a(X) \vee b(Y) &\leftarrow c(X, Y), \text{not } d(X) \\ &\leftarrow b(X), \text{not } c(X) \end{aligned}$$

The transformation of the second rule with the new relational symbol p is

$$\begin{aligned} c(X, Y) &\leftarrow a(X), p(X), p(Y) \\ c(X, Y) &\leftarrow b(Y), p(X), p(Y) \end{aligned}$$

It equals $\gamma(\Pi, p)$. The first and third rule are not considered. □

The following algorithm decides G-LSH(Π, r). It uses an oracle query $\Pi \models_b q$ which is true iff the ground atom q is an element of some stable model of the program Π. This is sometimes called *brave reasoning*. We call the corresponding problem the *brave reasoning problem* (BR).

Algorithm: A3
Input: program Π, rule $r \in \Pi$
Output: Boolean value b

1. Determine the active domain $\mathcal{D}_\Pi$.
2. Set $\mathcal{D}_0$ to $\mathcal{D}_\Pi$ and add new constants such that $|\mathcal{D}_0| = 4 \cdot |\mathcal{D}_\Pi|$.
3. Determine a new relational symbol p relative to Π.
4. Compute $A := \alpha(p, \mathcal{D}_0)$.
5. Compute $B := \gamma(\Pi, p)$.
6. For all $s, t \in H(r')$, $r' \in \mathrm{Gr}(r, \mathcal{D}_0)$:
7. If $s = t$ or $(A \cup B \cup \{s\} \models_b t$ and $A \cup B \cup \{t\} \models_b s)$ return $b :=$ false
8. Return $b :=$ true. □

Theorem 13. Algorithm A3 decides G-LSH.

Proof. We need only to proof that $A \cup B \cup \{s\} \models_b t$ iff t is reachable from s in $D^+(\mathrm{Gr}(\Pi, \mathcal{D}_0))$. The program $\Pi := A \cup B \cup \{s\}$ is Horn and has a stable model by construction. Therefore, Π has exactly one stable model I.

Every rule in B has the form

$$a(x_1, ..., x_n) \leftarrow b(y_1, ..., y_n), p(x_1), ..., p(x_n)$$

where the x_i and y_j are variables or constants. They need not to be different. The atom $p(x_i)$ is only there for making the rule safe. Note, that $p(c) \in I$ for every $c \in \mathcal{D}_0$ because $A \subseteq I$. The ground atom $a(c_1, ..., c_n)$ is in I for some $c_1, ..., c_n$ if $b(d_1, ..., d_n) \in I$ for some constants $d_1, ..., d_n$. From this it follows that for every atom $u \neq s$: $u \in I - A$ iff u is reachable from s in $D^+(\mathrm{Gr}(\Pi), \mathcal{D}_0)$. □

Algorithm A3 calls several times the oracle for the brave reasoning problem (BR). We now present an algorithm which directly reduces G-LSH to

BR. In the following x_1^n denotes the sequence $x_1, ..., x_n$ of variables or constants. We describe a relation $t(a, x_1^n, b, y_1^n)$ which solves the reachability problem first. We will need the following operators: The operator η assigns to $p \in \mathcal{R}^1$, $c \in \mathcal{D}$, $t \in \mathcal{R}^{2k}$, and to the two atoms $a(x_1^m)$ and $b(y_1^n)$ with $m, n \leq k-1$ the rule

$$t(a, x_1^m, u_1^{k-m}, b, y_1^n, v_1^{k-n}) \leftarrow \alpha(p, V_{\{x_1^m, y_1^n\}})$$

where $u_i = c$ for $1 \leq i \leq k-m-1$ and $v_i = c$ for $1 \leq i \leq k-n-1$.

Example 19. Let $a(x_1^m)$ be $r(X, 0, Y)$, $b(y_1^n)$ be $s(Z, Z)$, and k be 6. Then η generates the rule

$$t(a, X, 0, Y, c, c, b, Z, Z, c, c, c) \leftarrow p(X), p(Y), p(Z)$$

Note, that this rule is safe. □

Let r be a rule with $H(r) \neq \{\}$ and $B^+(r) \neq \{\}$. Define

$$\beta'(r, p, c, t) := \{\eta(p, c, t, a(x_1^m), b(y_1^n)) : a(x_1^m) \in H(r), b(y_1^n) \in B^+(r)\}$$

and

$$\gamma'(\Pi, p, c, t) := \bigcup \{\beta'(r, p, c, t) : r \in \Pi, H(r) \neq \{\}, B^+(r) \neq \{\}\}.$$

Example 20. Let Π be

$$\begin{aligned} a(1) & \\ a(X) \vee b(Y) &\leftarrow e(X, Y), \text{not } d(X) \\ &\leftarrow b(X), \text{not } d(X) \end{aligned}$$

The transformation of the second rule with the new relational symbol p is

$$\begin{aligned} t(a, X, c, e, X, Y) &\leftarrow p(X), p(Y) \\ t(b, Y, c, e, X, Y) &\leftarrow p(X), p(Y) \end{aligned}$$

It equals $\gamma'(\Pi, p, c, t)$. The first and third rule are not considered. □

Consider the rule

$$t(X_1^k, Z_1^k) \leftarrow t(X_1^k, Y_1^k), t(Y_1^k, Z_1^k)$$

together with $\gamma'(\Pi, p, c, t)$. This program describes a transitive closure.

To complete the reduction we need to select two ground atoms from the head of some instantiation of $r \in \Pi$. Therefore we need the following operator:

$$\delta(r, p, c, s) := \{\eta(p, c, s, a(x_1^m), b(y_1^n)) : a(x_1^m) \in H(r), b(y_1^n) \in H(r)\}$$

The following algorithm takes as input a program P and $r \in P$. It outputs a propositional atom b and a program Q such that $\overline{\text{G-LSH}}(P, r)$ holds iff $Q \models_b b$.

Algorithm: A4
Input: program P, rule $r \in P$
Output: propositional atom b, program Q

1. Determine the active domain $\mathcal{D}_P$.
2. Set $\mathcal{D}_0$ to $\mathcal{D}_P$ and add new constants such that $|\mathcal{D}_0| = 4 \cdot |\mathcal{D}_P|$.
3. Determine $k := \max_{ar}(P) + 1$.
4. Determine new and different relational symbols b, p, s, t relative to P: $b \in \mathcal{R}^0$, $p \in \mathcal{R}^1$, $s, t \in \mathcal{R}^k$.
5. Compute $A := \alpha(p, \mathcal{D}_0)$.
6. Fix a constant c.
7. Compute $B := \delta(r, p, c, s)$.
8. Compute $C := \gamma'(P, p, c, t)$.
9. Add $t(X_1^k, Z_1^k) \leftarrow t(X_1^k, Y_1^k), t(Y_1^k, Z_1^k)$ to C.
10. Add $b \leftarrow t(X_1^k, Y_1^k), t(Y_1^k, X_1^k), s(X_1^k, Y_1^k)$ to C.
11. Add $b \leftarrow s(X_1^k, X_1^k)$ to C.
12. Set $Q := A \cup B \cup C$.
13. Output (b, Q). □

Theorem 14. G-LSH is polynomial time reducible to BR.

Proof. We have to show that $\overline{\text{G-LSH}}(P, r)$ holds iff $Q \models_b b$. The set A consists of facts only. The relation p is used in B and C to make rules safe. The set C contains all rules for describing a transitive closure of the positive dependency graph D. The graph D is essentially $D^+(\text{Gr}(\Pi), \mathcal{D}_0)$ with the exception that every atom $a(x_1^l)$ with arity $l < k-1$ is replaced by an atom $a(x_1^l, c, ..., c)$ with arity $k-1$. The relation $t(a, x_1, ..., x_{k-1}, b, y_1, ..., y_{k-1})$ then holds in a stable model of Q iff $b(y_1^{k-1})$ is reachable from $a(x_1^{k-1})$ in D iff for the corresponding atoms $a(x_1^{l_1})$, $b(y_1^{l_2})$: $b(y_1^{l_2})$ is reachable from $a(x_1^{l_1})$ in $D^+(\text{Gr}(\Pi), \mathcal{D}_0)$. The relation s just contains every pair of atoms of some instantiation of $H(r)$ extended to arity k if necessary. Therefore, the rule

$$b \leftarrow t(X_1^k, Y_1^k), t(Y_1^k, X_1^k), s(X_1^k, Y_1^k),$$

expresses: if there exists $X_1, ..., X_k, Y_1, ..., X_k$ such that Y_1^k is reachable from X_1^k and vice versa, and X_1^k, Y_1^k are two atoms from the head of some instantiation of r then b is true. The atom b also becomes true if X_1^k are Y_1^k equal. The last rule added to c ensures this. Since these are the only possibilities to make b true $Q \models_b b$ holds iff $\overline{\text{G-LSH}}(P, r)$ holds. □

We will not give a detailed a analysis of the output size here. But it is easy that to see that the output size is quadratic in the input size.

Implementation Issues In the implementation of algorithm A2, A3, and A4 we use DLV syntax. This concerns the input, output, and the programs generated in algorithm A3. The supported DLV syntax in EBNF is the one we specified in the previous section. But DLV actually has a richer language. Especially, the variables can contain an underscore. This is not allowed here because we use it to simplify the algorithms A2, A3, and A4, i.e., the introduction of new symbols becomes easier. See the following example.

Example 21. Let r be

```
e(X,Y) v e(Y,Z):- e(Y,Z), g(Z,X), not e(X,X).
```

and s be

```
e(W,V) v e(V,V):- g(Z,V), not e(W,W).
```

For example, A might look like

```
{e(W,V), e(V,V), g_p(Z,V), e_n(W,W)}
```

and B like

$$\{\texttt{e(c\_1,c\_2), e(c\_2,c\_3), e(c\_1,c\_1),}$$
$$\texttt{e\_p(c\_2,c\_3), g\_p(c\_3,c\_1), e\_n(c\_1,c\_1)}\}.$$

The relational symbols `e_p`, `g_p`, and `e_n` are new symbols because the underscore _ does not occur in the supported DLV syntax. The same holds for the constants `c_1`,`c_2`,`c_3`. □

The algorithms A2, A3, and A4 were implemented in the programming language Perl. In practice it is the case that algorithm A4 outperforms algorithm A3.

The toolbox we call Simplifier consists of two scripts which take as input a program P in DLV syntax. The first is named `simplify_subs`. After calling it with

```
simplify_subs input.dl
```

on the command line it outputs all rules $r, s \in P$ such that SUBS$^*(r, s)$ holds. The file `input.dl` is just the input DLV file. If no input file is specified then the input is read from the standard input device. The second script is named `simplify_glsh`. After calling it with

```
simplify_glsh input.dl
```

on the command line it outputs all rules $r \in P$ such that G-LSH(P, r) holds.

Chapter 6

Further Work

In this chapter we discuss our work and open questions. We also discuss some related work.

We introduced several techniques for program simplification and optimization of disjunctive datalog programs with negation. Our approach was based upon single results in the propositional case. Lin and Chen [LC05] tried a more systematic way; also in the propositional case. They concentrated on strong equivalence and studied the k-m-n simplification problem for fixed k, m, n: Is $\{a_1, ..., a_k, b_1, ..., b_m\}$ strongly equivalent to $\{a_1, ..., a_k, c_1, ..., c_n\}$ where a_i, b_j, c_k are rules? Note that for variable k, m, n this problem would be just the problem of deciding strong equivalence of arbitrary programs. For $m = 1$, $n = 0$ we have what we called rule elimination. It is remarkable that Lin and Chen found full characterizations for the 0-1-0 and 1-1-0 problem and others. Whereas we only showed that if for some rule r, TAUT(r) or CONTRA(r) holds, r can be eliminated they also showed that in the propositional case the converse holds, i.e., if the rule can be eliminated by considering only the rule itself and not the rest of the program then TAUT(r) or CONTRA(r) hold. We expect that results of this kind also hold for non-ground programs.

Another question is to find further techniques for simplification which are useful in practice, i.e., they indeed simplify/optimize programs used in practice and their complexity is low. Consider the case whether we find a technique for the 2-2-1 simplification problem of non-ground programs. Does such a technique often apply in practice? And what about the complexity of the general 2-2-1 simplification problem? The simplification problems we

studied are elementary and intuitively it seems reasonable to assume that they apply to programs used in practice, especially if the programs are generated automatically. But an important topic for further work is to apply the results of program simplification and optimization to a larger class of programs than we did in our experiments. We also remark that our concept of rule transformation does not fit into the approach of Lin and Chen since in testing if a rule can be transformed we had to check for head-cycle freeness, i.e., the whole program needed to be considered. Therefore we can not fix the parameter k here.

If we look at the correctness proofs of the techniques we see some similarities. One might question if there are some general principles behind it, and indeed, there are. We will not go into the details because for explaining it we would need the formal definitions. But roughly speaking, we can find "replacement schemes" and study them in general. A technique then is an instance of such a replacement scheme. Together with these generalizations our results of Chapters 3 and 4 will be published in an upcoming paper.

At the end we will discuss our implementations. Our first implementation solves the general strong equivalence problem. It was published in [EFT05]. There one also finds some experiments. This implementation can be used for program simplification and optimization. But because of the high complexity this will not work for large, complicated programs or even for online optimization in general. (But for small programs one should consider this implementation.) The same holds for the Simplifier toolbox. With this considerations the following question is immediate at hand: How to improve the implementations? Deciding SUBS* was reduced to the Boolean conjunctive query problem. This problem is closely related to the constraint satisfaction problem and some others. These problems are well studied and many special cases were identified. For example, see [GLS02] by Gottlob et al. This knowledge might be used in further work. We also solved the general problem of deciding G-LSH. However, it is not clear if there exists an algorithm for, say, G-LSH$_3$ which runs in time $O(n^3)$. These issues remain for further research.

Chapter 7

Conclusion

We contributed to the theory and practice of program simplification and optimization of non-ground disjunctive datalog programs with negation by introducing techniques for rule elimination (G-TAUT, G-CONTRA, G-RED$^-$, G-NONMIN, G-S-IMPL, G-SUBS) and for rule transformation (G-LSH, G-LSH$_k$). Similar techniques were studied for the propositional case in [ONA01] and [EFTW04]. We proved the correctness of these techniques. G-TAUT, G-CONTRA, G-RED$^-$, G-NONMIN, G-S-IMPL, and G-SUBS preserve strong and uniform equivalence. G-LSH and G-LSH$_k$ preserve uniform equivalence. These results establish that we can use the considered techniques for program simplification and optimization in principle. Of course, the actual use of these techniques depends on the complexity of their applicability. The complexity results we established are summarized in the following table.

Technique	Complexity
G-TAUT	in P
G-CONTRA	in P
G-RED$^-$	in P
G-NONMIN	NP complete
G-S-IMPL	NP complete
G-SUBS	NP complete
G-LSH	PSPACE complete
G-LSH$_k$	NL complete

G-TAUT, G-CONTRA, and G-RED$^-$ are efficiently decidable and easy to implement. Therefore we concentrated on the remaining techniques.

G-SUBS is a generalization of G-NONMIN and G-S-IMPL with the same complexity. Hence, G-SUBS might be used instead of G-NONMIN and G-S-IMPL. Algorithm A2 reduces G-SUBS to the Boolean conjunctive query problem. The implementation of A2 is written in Perl and uses the DLV system.

G-LSH$_k$ is decidable in polynomial time since NL $\subseteq$ P. But the running times of algorithm A3 and A4 crucial depend on k. Indeed, an upper bound for the time complexity of G-LSH$_k$ is $O(n^{c \cdot k})$ for some $c \in \mathbb{N}$. Algorithm A3 decides G-LSH with several oracle calls and A4 reduces G-LSH to an evaluation problem of disjunctive datalog problems with negation. Empirical results show that algorithm A4 outperforms A3 and hence algorithm A4 should be used instead of A3. The implementation of A4 is again written in Perl and uses the DLV system.

We also implemented an algorithm for deciding strong equivalence of two programs. Algorithm A1 reduces the problem to the unsatisfiability problem of Bernays-Schönfinkel formulas. The implementation of algorithm A1 is written in C++ and uses the automated theorem prover Darwin.

To summarize, the results in this thesis provide a first step towards the development of effective optimization methods for non-ground Answer Set Programming, an issue which has been merely addressed for propositional programs so far.

Bibliography

[AKL01] C. Anger, K. Konczak, T. Linke. NoMoRe: A System for Non-Monotonic Reasoning, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, LNCS 2173, pp. 406-410, 2001.

[Baral02] C. Baral. Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, 2003.

[BD99] S. Brass, J. Dix. Semantics of (Disjunctive) Logic Programs Based on Partial Evaluation. *Journal of Logic Programming*, vol. 38(3), pp. 167-213, 1999.

[BT03] P. Baumgartner, C. Tinelli. The Model Evolution Calculus, *Fachberichte Informatik 1-2003*, Universität Koblenz-Landau, Institut für Informatik, 2003.

[D+01] T. Dell'Armi, W. Faber, G. Ielpa, C. Koch, N. Leone, S. Perri, G. Pfeifer. System Description: DLV. *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, LNCS 2173, pp. 424-428, Springer, 2001.

[DE+01] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov. Complexity and expressive power of logic programming, *ACM Computing Surveys*, pp. 374-425, ACM Press, 2001.

[E+04] T. Eiter, W. Faber, M. Fink, G. Greco, D. Lembo, H. Tompits. Methods and Techniques for Query Optimization, *Infomix Consortium, Technical Report D5.3* (2004).

[E+06] T. Eiter, M. Fink, H. Tompits, P. Traxler, S. Woltran. Replacements in Non-Ground Answer-Set Programming. Accepted for publication in the *Proceedings of the 20th Workshop on Logic Programming (WLP)*, 2006.

[EF03] T. Eiter, M. Fink. Uniform Equivalence of Logic Programs under the Stable Model Semantics, *Proceedings of the 19th International Conference on Logic Programming*, LNCS 2916, pp. 224-238, Springer, 2003.

[EF+04] T. Eiter, W. Faber, M. Fink, G. Pfeifer, S. Woltran. Complexity of Model Checking and Bounded Predicate Arities for Non-ground Answer Set Programming, *Proceedings of the 9th International Conference on Principles of Knowledge Representation and Reasoning*, pp. 377-387, 2004.

[EFT05] T. Eiter, W. Faber, P. Traxler. Testing Strong Equivalence of Nonmonotonic Datalog Programs - Implementation and Examples, *Proceedings of the 8th International Conference on Logic Programming and Non Monotonic Reasoning*, LNCS 3662, pp. 437-441, Springer, 2005.

[EFTW04] T. Eiter, M. Fink, H. Tompits, S. Woltran. Simplifying Logic Programs under Uniform and Strong Equivalence, *Proceedings of the 7th International Conference on Logic Programming and Nonmonotonic Reasoning*, LNCS 2923, pp. 87-99, Springer, 2004.

[EFTW05] T. Eiter, M. Fink, H. Tompits, S. Woltran. Strong and Uniform Equivalence in Answer-Set Programming: Characterizations and Complexity Results for the Non-Ground Case, *Proceedings of the 20th National Conference on Artificial Intelligence*, pp. 695-700, AAAI Press, 2005.

[EFW05] T. Eiter, M. Fink, S. Woltran. Semantical Characterizations and Complexity of Equivalencs in Answer Set Programming, *Institute of Information Systems, Vienna University of Technology, Infsys Research Report 1843-05-01*, 2005.

[EGM97] T. Eiter, G. Gottlob, H. Mannila. Disjunctive Datalog, *ACM Transactions on Database Systems*, pp. 364-418, ACM Press, 1997.

[FL05] P. Ferraris, V. Lifschitz. Mathematical Foundations of Answer Set Programming. We Will Show Them!. In *Essays in Honour of Dov Gabbay*, vol. 1, pp. 615-664, College Publications, 2005.

[GJ79] M. R. Garey, D. S. Johnson. Computers and Intractability. W. H. Freeman, 1979.

[GL91] M. Gelfond, V. Lifschitz. Classical negation in logic programs and disjunctive databases, *New Generation Computing*, pp. 365-385, 1991.

[GL02] M. Gelfond, N. Leone. Logic Programming and Knowledge Representation - The A-Prolog Perspective. *Artificial Intelligence*, vol. 138, pp. 3-38, 2002.

[GLS02] G. Gottlob, N. Leone, F. Scarcello. Hypertree Decompositions and Tractable Queries. *Journal of Computer and System Science*, vol. 64(3), pp. 579-627, 2002.

[Johnson90] D. S. Johnson. A Catalog of Complexity Classes. In *Handbook of Theoretical Computer Science: Volume A, Algorithms and Complexity*, pp. 67-161, MIT Press, 1990.

[L+05] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello. The DLV System for Knowledge Representation and Reasoning, *ACM Transactions on Computational Logic*, 2005. To appear.

[LAK02] T. Linke, C. Anger, K. Konczak. More on NoMoRe. *Proceedings of the 8th European Conference on Artificial Intelligence*, LNAI 2424, pp. 468-480, Springer, 2002.

[LC05] F. Lin, Y. Chen. Discovering Classes of Strongly Equivalent Logic Programs, *Proceedings of the 19th International Joint Conference of Artificial Intelligence*, 2005.

[Lierler05] Y. Lierler. cmodels - SAT-Based Disjunctive Answer Set Solver. *Proceedings of the 8th International Conference on Logic Programming and Non Monotonic Reasoning*, LNCS 3662, pp. 447-451, Springer, 2005.

[Lin02] F. Lin. Reducing Strong Equivalence of Logic Programs to Entailment in Classical Propositional Logic, *Proceedings of the Eight International Conference on Principles of Knowledge Representation and Reasoning*, pp. 170-176, Morgan Kaufmann, 2002.

[LPV01] V. Lifschitz, D. Pearce, A. Valverde. Strongly Equivalent Logic Programs, *ACM Transactions on Computational Logic*, pp. 526-541, ACM Press, 2001.

[LZ02] F. Lin, Y. Zhao. ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers, *Proceedings of the 18th National Conference on Artificial Intelligence*, pp. 112-117, AAAI, 2002.

[LZ03] F. Lin, J. Zhao. On Tight Logic Programs and Yet Another Translation from Normal Logic Programs to Propositional Logic. *Proceedings*

of the 18th International Joint Conference on Artificial Intelligence, pp. 853-858, Morgan Kaufmann, 2003.

[Maher88] M. J. Maher. Equivalences of logic programs. In *Foundations of Deductive Databases and Logic Programming*, pp. 627-658, Morgan Kaufmann, 1988.

[MT99] W. Marek, M. Truszczyński. Stable Models and an Alternative Logic Programming Paradigm. In *The Logic Programming Paradigm: A 25-Year Perspective*, pp. 375-398, Springer, 1999.

[Niemelae99] I. Niemelä. Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm. *Annals of Mathematics and Artificial Intelligence*, vol. 25, pp. 3-4, 1999.

[NSS00] I. Niemelä, P. Simons, T. Syrjänen. Smodels: A System for Answer Set Programming, *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*, 2000.

[NSS02] I. Niemelä, P. Simons, T. Soininen. Extending and Implementing the Stable Model Semantics, *Artificial Intelligence*, pp. 181-234, Elsevier Science Publishers, 2002.

[ONA01] M. Osorio, J. Navarro, J. Arrazola. Equivalence in Answer Set Programming, *Proceedings of the 11th International Workshop on Logic Based Program Synthesis and Transformation*, LNCS 2372, pp. 57-75, Springer, 2001.

[Pap94] C. Papadimitriou. Computational Complexity. Addison-Wesley, 1994.

[PTW01] D. Pearce, H. Tompits, S. Woltran. Encodings for Equilibrium Logic and Logic Programs with Nested Expressions, *Proceedings of the 10th Portuguese Conference on Artificial Intelligence*, LNCS 2258, pp. 306-320, Springer, 2001.

[Sagiv88] Y. Sagiv. Optimizing datalog programs. In *Foundations of Deductive Databases and Logic Programming*, pp. 659-698, Morgan Kaufmann, 1988.

[S76] L. J. Stockmeyer. The Polynomial-Time Hierarchy. *Theoretical Computer Science*, vol. 3(1), pp. 1-22, 1976.

[Sipser97] M. Sipser. Introduction to the Theory of Computation. PWS Publishing Company, 1997.

[Turner01] H. Turner. Strong Equivalence for Logic Programs and Default Theories (Made Easy). *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, LNCS 2173, pp. 81-92, 2001.

[Turner03] H. Turner. Strong Equivalence Made Easy: Nested Expressions and Weight Constraints. *Theory and Practice of Logic Programming*, pp. 609-622, 2003.

[WZ05] K. Wang, L. Zhou. Comparisons and Computation of Well-founded Semantics for Disjunctive Logic Programs. *ACM Transactions on Computational Logic*, vol. 6(2), pp. 295-327, ACM Press, 2005.

Appendix A

Unsafe Programs

In the absence of the safety requirement we use the additional requirement $\mathcal{D}_{\Pi-\{r\}} \subseteq \mathcal{D}_\Pi$, i.e., the active domain of Π does not change if r is removed. Denote this requirement by AD.

Proposition 16. (Correctness) Let Π be a (not necessarily safe) program and $r \in \Pi$. $\Pi \equiv_s \Pi - \{r\}$ if

1. AD + G-TAUT(r),
2. AD + G-CONTRA(r),
3. AD + G-NONMIN(Π, r, s),
4. AD + G-S-IMPL(Π, r, s), or
5. AD + G-SUBS(Π, r, s).

Proof. 1) $\Pi \equiv_s \Pi - \{r\}$ iff for all programs R: $\Pi \cup R \equiv \Pi - \{r\} \cup R$. Define $G := \mathrm{Gr}(\Pi \cup R)$, $G' := \mathrm{Gr}(\Pi - \{r\} \cup R)$, and $\{r'_1, ..., r'_n\} := G - G'$. It follows from G-TAUT(r) that TAUT($\mu(r'_1)$). Therefore $\mu(G) \equiv \mu(G) - \{\mu(r'_1)\}$ because of Proposition 5. And since μ preserves ordinary equivalence it holds that $G \equiv G' - \{r'_1\}$. By repeating this argument it follows that $G \equiv G - \{r'_1, ..., r'_n\} = G'$ which implies $\Pi \cup R \equiv \Pi - \{r\} \cup R$ since the active domain does not change. Uniform equivalence is implied by strong equivalence.

2, 3, 4, 5) Analogous. □