Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 85
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 85
Espoo 2003 HUT-TCS-A85

TESTING THE EQUIVALENCE OF DISJUNCTIVE LOGIC PROGRAMS

Emilia Oikarinen

TEKNILLINEN KORKEAKOULU
TEKNISKA HÖGSKOLAN
HELSINKI UNIVERSITY OF TECHNOLOGY
TECHNISCHE UNIVERSITÄT HELSINKI
UNIVERSITE DE TECHNOLOGIE D'HELSINKI

Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 85
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 85
Espoo 2003 HUT-TCS-A85

TESTING THE EQUIVALENCE OF DISJUNCTIVE LOGIC PROGRAMS

Emilia Oikarinen

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science

Teknillinen korkeakoulu
Tietotekniikan osasto
Tietojenkäsittelyteorian laboratorio

Distribution:
Helsinki University of Technology
Laboratory for Theoretical Computer Science
P.O.Box 5400
FIN-02015 HUT
Tel. +358-0-451 1
Fax. +358-0-451 3369
E-mail: lab@tcs.hut.fi

ISBN 951-22-6878-7
ISSN 1457-7615

Multiprint Oy
Helsinki 2003

ABSTRACT: To solve a problem in answer set programming (ASP), one constructs a logic program so that its answer sets correspond to the solutions of the problem and computes the answer sets of the program using a special purpose search engine. The encodings are not unique, i.e. several versions of a program can be used e.g. in optimizing the execution time or space. Since the solutions to the problem correspond to answer sets of the program, it is necessary to ensure that the different encodings yield the same output. In ASP this means that one has to check whether given two logic programs have the same answer sets, i.e., whether they are semantically equivalent. We consider in this work the class of disjunctive logic programs, that form a proper generalization of normal logic programs.

The equivalence of logic programs P and Q can naturally be verified using an explicit cross-check of all the answer sets of both programs. Our aim is, however, to develop a systematic method for testing the equivalence so that a naive cross-check of answer sets is not needed. The idea is to translate logic programs P and Q into a single logic program that has an answer set if and only if P has an answer set that is not an answer set of Q. Thus answer sets of the translation (if found), act as a counter-examples for the equivalence of P and Q. The counter-examples for equivalence divide naturally in two types. Thus the search for counter-examples can be performed separately for both types using a two-phased translation.

We have implemented the translation functions. Experiments with the implementation show that in several cases the translation-based approach is superior to the naive cross-checking approach, especially, if the programs to be tested have several answer sets and are likely to be nonequivalent. If the number of answer sets is low, then the naive cross-checking approach is likely to be faster. Furthermore, in general it is faster to use the two-phased translation.

KEYWORDS: disjunctive logic programs, equivalence testing, stable model semantics, answer set programming, non-monotonic reasoning, computational complexity

CONTENTS

List of Figures

NOTATIONAL CONVENTIONS AND ABBREVIATIONS

$\subseteq$	subset
$\subset$	proper subset
$\cup$	union
$\cap$	intersection
$\setminus$	set minus
$\models$	satisfaction relation
$\equiv$	equivalence relation
$\equiv_s$	strong equivalence relation
$\emptyset$	empty set
$\bot$	an atom that is always unsatisfied
$\neg$	classical negation
$\sim$	negation as failure (to prove), default negation
Σ^*	set of all strings over an alphabet Σ
$\lvert \cdot \rvert$	size of a set
2^X	$\{Y \mid Y \subseteq X\}$, power set
$P, Q, R, \ldots$	logic programs
$A, B, C, \ldots$	sets of atoms, interpreted conjunctively
$\bigvee A, \bigvee B, \bigvee C, \ldots$	sets of atoms, interpreted disjunctively
$a, a_1, \ldots, a_n, b, \ldots$	atoms
$\sim A$	$\{\sim a \mid a \in A\}$
$A^\bullet$	$\{a^\bullet \mid a \in A\}$
A°	$\{a^\circ \mid a \in A\}$
$A \leftarrow B, \sim C$	a (disjunctive) rule
$\mathrm{Hb}(\cdot)$	Herbrand base
$\mathrm{Cn}(\cdot)$	set of logical consequences
$\mathbf{SM}(\cdot)$	set of stable models
$\mathbf{MM}(\cdot)$	set of minimal models
$\mathrm{LM}(\cdot)$	least model
$\mathbf{SuppM}(\cdot)$	set of supported models
P_M	Gelfond-Lifschitz reduct of P w.r.t. $M \subseteq \mathrm{Hb}(P)$
EXT	extension function
PROJ	projection function
$\mathrm{CE}(\cdot, \cdot)$	set of counter-examples
$\mathrm{CE}_1(\cdot, \cdot)$	set of counter-examples of type T1
$\mathrm{CE}_2(\cdot, \cdot)$	set of counter-examples of type T2
$\mathrm{TR}(\cdot, \cdot)$	translation for equivalence testing
$[\mathrm{TR}_1(\cdot, \cdot), \mathrm{TR}_2(\cdot, \cdot)]$	two-phased translation for equivalence testing
$\mathrm{EG}(\cdot)$	translation from a QSAT instance to a disjunctive logic program
$\mathrm{J}(\cdot)$	translation from a QSAT instance to a disjunctive logic program

SM	$\{P \mid \exists M \subseteq \mathrm{Hb}(P)$ such that $M \in \mathbf{SM}(P)\}$
NotMin	$\{(P, M) \mid$ there exists $M' \subset M$ s.t. $M' \models P\}$
IMPR	$\{(P, Q) \mid \mathbf{SM}(P) \subseteq \mathbf{SM}(Q)\}$
IMPL	$\{(P, Q) \mid (Q, P) \in \mathrm{IMPR}\}$
EQV	$\{(P, Q) \mid P \equiv Q\}$
$\mathbf{P}$	time-complexity class containing languages decidable in polynomial time using a deterministic Turing machine
$\mathbf{NP}$	time-complexity class containing languages decidable in polynomial time using a nondeterministic Turing machine
$\mathbf{coNP}$	time-complexity class containing languages the complement of which belongs to $\mathbf{NP}$
$\mathbf{P}^{\Sigma_k^{\mathbf{P}}} = \Delta_{k+1}^{\mathbf{P}}$	time-complexity class containing languages decidable in polynomial time using a deterministic $\Sigma_k^{\mathbf{P}}$-oracle Turing machine
$\mathbf{NP}^{\Sigma_k^{\mathbf{P}}} = \Sigma_{k+1}^{\mathbf{P}}$	time-complexity class containing languages decidable in polynomial time using a nondeterministic $\Sigma_k^{\mathbf{P}}$-oracle Turing machine
$\mathbf{coNP}^{\Sigma_k^{\mathbf{P}}} = \Pi_{k+1}^{\mathbf{P}}$	time-complexity class containing languages the complement of which belongs to $\mathbf{NP}^{\Sigma_k^{\mathbf{P}}}$
DLP	disjunctive logic program
ASP	answer set programming
HT	logic of here-and-there
CNF	conjunctive normal form
DNF	disjunctive normal form
SAT	propositional satisfiability problem
QSAT	quantified propositional satisfiability problem
GNT	solver for disjunctive logic programs
DLV	solver for disjunctive logic programs
SMODELS	solver for weight constraint logic programs
LPARSE	front-end of SMODELS
LPEQ	translator for testing the equivalence of weight constraint logic programs
DLPEQ	translator for testing the equivalence of disjunctive logic programs

1 INTRODUCTION

To solve a problem in answer set programming (ASP) [28, 29], one needs to construct a logic program so that its answer sets correspond to the solutions of the problem at hand. Answer sets of the program are then computed using a special purpose search engine. Answer set programming paradigm was formally identified as a self-standing programming paradigm in the late 1990s. It has since been the subject of increasing attention. As efficient search engines such as DLV [6], SMODELS [37], and GNT [18, 19], have been developed, ASP applications have emerged in growing numbers. Areas in which ASP has been recently used include for example planning [21, 29], product configuration [38], computer aided verification [13], wire routing in VLSI design [7] and logical cryptanalysis [14].

The answer set programming paradigm has its roots in traditional logic programming (i.e. in languages such as Prolog for example). Both paradigms are rule-based and declarative. The basic syntax of rules in ASP is similar to that of traditional logic programming but function symbols are not allowed. In ASP the semantics, i.e. how a logic program is actually interpreted, is totally different from traditional logic programming. In ASP the programs are interpreted as sets of constraints for answer sets to be computed whereas in traditional logic programming the rules are used to model recursive definitions.

A *normal* or *general* logic program [25], in which the *negation as failure* [2] is used, is the standard form of logic programs in ASP. There are a number of generalizations to this standard syntax: classical negation [10], disjunction in the heads of rules [11, 35], default negation in the heads of disjunctive rules [15], nested rules [23], and weight rules [37]. The semantics for the more general classes of logic programs is defined using the respective generalizations of the *stable model semantics* [9], which is the standard semantics in ASP. As discussed in [10], the above mentioned *answer sets* are consistent sets of classical literals that are stable in the same sense as stable models [9]. In this work, we concentrate on disjunctive logic programs. Disjunctive logic programs cover the generalizations of normal logic programs mentioned above — either as a proper generalization or through translations [10, 15, 16, 33, 8]. Thus a wide variety of logic programs is implicitly under consideration in this work.

Due to the declarative nature of ASP, it is an appealing approach to solving a variety of problems. There are, however, some difficulties regarding the development of programs using the ASP paradigm. Encodings for a particular problem are by no means unique. Thus, despite the declarative nature of ASP, the development of programs resembles that of programs in conventional programming; the programmer often develops several versions of the program used to solve the problem at hand, e.g., when optimizing the execution time and space. Since the solutions to the problem correspond to answer sets of the program, it is necessary to ensure that the different encodings yield the same output. In ASP this means that one needs to check whether two given logic programs P and Q have the same answer sets, i.e., P and Q are semantically equivalent. This corresponds to the weaker notion of equiva-

lence addressed by Lifschitz, Pearce and Valverde [22]. Lifschitz, Pearce and Valverde also present a stronger notion of equivalence called strong equivalence: P and Q are *strongly equivalent* if P and Q yield same answer sets in every possible context in which they can be placed, that is, P and Q are equivalent as parts of any other programs. We concentrate on the weaker notion of equivalence in this work.

The equivalence of two given logic programs can naturally be tested using an explicit cross-check of all answer sets of both programs. If the programs to be tested have many answer sets, a naive approach easily becomes infeasible and thus a different approach might be more efficient. Our aim is to develop a systematic method for testing the equivalence so that an explicit cross-check of answer sets is not needed. We assume that such a systematic method is likely to be faster than the naive cross-checking approach.

Lin [24] and Turner [40] have independently shown that testing the *strong equivalence* of disjunctive logic programs is **coNP**-complete. Turner [40] also shows that testing the weak equivalence is $\Pi_2^{\mathbf{P}}$-hard. We show that the problem is indeed $\Pi_2^{\mathbf{P}}$-complete. The problem of finding stable models for a disjunctive logic program is also $\Pi_2^{\mathbf{P}}$-complete [5]. Thus the computational complexity of equivalence testing is the same as the complexity of finding stable models. Therefore it is possible to construct a polynomial-time transformation reducing equivalence testing to the problem of finding stable models for a disjunctive logic program.

Our aim is to develop a polynomial transformation from two disjunctive programs P and Q to a single program $\mathrm{TR}(P, Q)$ in such a way that the program $\mathrm{TR}(P, Q)$ has an answer set if and only if P and Q are not equivalent. Answer sets of the translation (if found) can then be seen to represent *counter-examples* to the assumed equivalence of P and Q. However, a polynomial-time reduction preserves only the yes/no answers of the decision problem. The reduction itself can be arbitrary as long as it is computable in polynomial time. Therefore the usefulness of such a transformation is not automatically clear. We believe that a systematic translation can be formed in such a manner that the answer sets of the translation can be used to extract the actual counter-examples for the equivalence of P and Q.

As a matter of fact, a systematic method for testing the equivalence of logic programs has been developed [20], but only in the case of weight constraint programs [37] supported by the SMODELS system. The idea in this approach is similar to the above mentioned translation-based one: P and Q are translated into a single logic program $\mathrm{EQT}(P, Q)$ that has an answer set if and only if P has an answer set that is not an answer set of Q. Thus the equivalence of P and Q can be established by showing that $\mathrm{EQT}(P, Q)$ and $\mathrm{EQT}(Q, P)$ have no answer sets. Using the translation one can make use of the existing SMODELS system for the search of counter-examples. Thus there is no need for a special purpose search engine for equivalence testing in this approach. Only a translator, such as LPEQ [17], needs to be implemented.

To summarize, our aim in this work is to generalize the translation-based method to the disjunctive case so that dedicated search engines such as GNT [18] or DLV [6] can be used for actual computations. We believe that as in [20], a systematic translation will be more efficient than the naive cross-check of answer sets of the programs.

The rest of this work is organized as follows.

- Chapter 2. First, we introduce the syntax of disjunctive logic programs. We continue our discussion with the stable model semantics and end this chapter by pointing out some useful properties of stable models.

- Chapter 3. We discuss the formal definitions of equivalence relations for disjunctive logic programs distinguishing two types of counter-examples. We also study the computational complexity of the equivalence testing problem in the disjunctive case, giving a proof of its $\Pi_2^{\mathbf{P}}$-completeness.

- Chapter 4. We present two translation-based techniques for testing the equivalence of disjunctive programs. One consists of a one-shot translation that aims to capture both types of counter-examples at once. The other technique employs two translations which capture the two types of counter-examples separately. We also address the correctness of the translations establishing that there is a very tight one-to-one correspondence between counter-examples and the stable models of the translations.

- Chapter 5. We report results from our experiments with a prototype implementation, a translator called DLPEQ, based on the two translations described in Chapter 4. The design of DLPEQ is compatible with GNT and DLV so that equivalence testing is enabled in practice. The experiments compare the performance of the two translations with a naive approach involving an explicit cross-check of the stable models of both programs.

- Chapter 6. We present our conclusions and future work.

2 DISJUNCTIVE LOGIC PROGRAMS

In this chapter we define disjunctive logic programs in the propositional case[1]. First, we introduce the syntax of disjunctive logic programs in Section 2.1. In Section 2.2 we restrict to positive programs and consider formal semantics in that restricted case. In Section 2.3 we consider the whole class of disjunctive logic programs introducing the stable model semantics. We end this chapter with useful properties of stable models in Section 2.4.

2.1 SYNTAX

We use the symbol "$\sim$" to denote *default negation* or *negation as failure to prove* [2] to distinguish it from classical negation "$\neg$". Default negation differs from the classical one, as in 'negation as failure' we conclude $\sim a$, if we cannot prove that a is true. The following example will illustrate this difference.

Example 2.1. Consider a proposition $a \vee b$ claiming that a or b is true. We cannot prove that a is true (since $a \vee b$ can be true even if a is not true). Thus we conclude $\sim a$ which can be interpreted as "there is no reason to assume that a is true". On the other hand, we cannot conclude $\neg a$, since we cannot prove that a is not true, i.e. a is false. ■

A default literal is an atom a or its default negation $\sim a$. We define the set of negative literals as $\sim A = \{\sim a \mid a \in A\}$ for any set of atoms A.

Definition 2.2. A rule in disjunctive logic programming is an expression of the form

$$a_1 \mid \ldots \mid a_n \leftarrow b_1, \ldots, b_m, \sim c_1, \ldots, \sim c_k, \tag{2.1}$$

where $a_1, \ldots, a_n, b_1, \ldots, b_m, c_1, \ldots, c_k$ are atoms and n, m and k are non-negative integers.

A rule consists of two parts: the *head* $a_1 \mid \ldots \mid a_n$ and the rest of the rule called the *body*. The atoms in the head are interpreted disjunctively, i.e. $\mid$ denotes disjunction and the literals in the body are interpreted conjunctively. Intuitively, a rule of the form (2.1) is used as an *inference rule*, i.e. any of the head atoms can be inferred given that all the literals in the body can be inferred. The order of literals in a rule is not significant. Thus we can use a shorthand $A \leftarrow B, \sim C$ for a rule, where A, B and C are sets ($|A| = n$, $|B| = m$ and $|C| = k$). If the head of a rule is empty, the rule is called an *integrity constraint*. Intuitively, all the literals in the body of an integrity constraint cannot be inferred. If all the literals in the body of an integrity constraint could be inferred, we should conclude that there exists an atom in the head (which is empty) that can be inferred, which is a contradiction.

A *disjunctive logic program (DLP)* P is a set of rules of the form (2.1). The *Herbrand base* of P, $\mathrm{Hb}(P)$, is the set of all atoms appearing in P. If

[1] Disjunctive programs with variables can be covered using Herbrand instantiation thus transforming the program into a grounded, variable-free, one.

$n = 1$ for each rule in P, then P is a *disjunction-free* or *normal program*. If $k = 0$ for each rule in P, then P is a *positive* logic program.

Example 2.3. Let us consider the following disjunctive logic program P.

$$
\begin{aligned}
&a \mid b \mid c.\\
&\bot \leftarrow a, {\sim}b.\\
&e \leftarrow b, {\sim}c, d.\\
&a \leftarrow e, d.\\
&a \mid d \leftarrow {\sim}b
\end{aligned}
$$

Let us introduce some notational conventions used in this work. The rules are separated with full stops and the symbol "$\leftarrow$" is dropped if the body of the rule is empty (e.g. the first rule in this example). The second rule is an example of an integrity constraint. The empty head is denoted with the symbol "$\bot$" (an atom that is always unsatisfied). ■

2.2 FORMAL SEMANTICS

In the previous section we defined the syntax of disjunctive logic programs and discussed the intuition behind the semantics. In this and the following section we present a model-theoretic semantics corresponding to this intuition. The aim is to formally define when a particular atom is inferable and when not, given a set of rules. We begin our discussion with the concept of *classical model* defined through the classical satisfaction relation and then consider the concept of logical consequences of a program.

Given a disjunctive logic program P, an *interpretation* I is a subset of $\mathrm{Hb}(P)$. An interpretation defines which atoms in $\mathrm{Hb}(P)$ are true ($a \in I$) and which are false ($a \notin I$). The satisfaction relation $I \models r$ is defined for default literals and rules as follows.

Definition 2.4. Given a disjunctive logic program P and an interpretation $I \subseteq \mathrm{Hb}(P)$,

- an atom a is satisfied in I ($I \models a$) if and only if $a \in I$,
- a negative literal ${\sim}a$ is satisfied in I ($I \models {\sim}a$) if and only if $I \not\models a$ ($a \notin I$),
- a set of literals L is satisfied in I ($I \models L$) if and only if $I \models l$ for each literal $l \in L$,
- a disjunctive set of literals $\bigvee L$ is satisfied in I ($I \models \bigvee L$) if and only if there exists a literal $l \in L$ such that $I \models l$,
- a rule $A \leftarrow B, {\sim}C$ is satisfied in I if and only if $I \models B \cup {\sim}C$ implies $I \models \bigvee A$.

Definition 2.5. An interpretation I is a *(classical) model* of program P (denoted by $I \models P$) if and only if $I \models r$ for each rule $r \in P$.

Let us consider Definition 2.4 in the case $L = \emptyset$. Given an arbitrary interpretation I, it holds that $I \models \emptyset$ and $I \not\models \bigvee \emptyset$. Since the body of a rule is

interpreted conjunctively, a rule with empty body is satisfied if and only if its head is satisfied. The head of a rule is interpreted disjunctively. An integrity constraint is thus satisfied if and only if its body is not satisfied. Since we use $\bot$ to denote an empty head, we have $I \not\models \bot$ for all interpretations I.

Classical models do not correspond to our intuitive interpretation of rules as inference rules, since a rule is satisfied in an interpretation I even if only its head it satisfied. Thus we cannot use semantics based on classical models as such.

For now, let us restrict our considerations to *positive normal* logic programs consisting of rules of the form

$$a \leftarrow b_1, \ldots, b_m,$$

where a and $b_1, \ldots, b_m$ are atoms and $m \geq 0$ is an integer. We define the logical consequences of a program P in the standard way.

Definition 2.6. An atom $a \in \mathrm{Hb}(P)$ is a logical consequence of a logic program P ($P \models a$) if and only if $a \in M$ for each interpretation $M \subseteq \mathrm{Hb}(P)$ such that $M \models P$. The set of logical consequences of P is $\mathrm{Cn}(P) = \{a \in \mathrm{Hb}(P) \mid P \models a\}$.

Thus an atom belongs to the set of logical consequences of a program if and only it is necessary to include it into each model of the program.

Example 2.7. Consider a positive normal program $P = \{a \leftarrow b\}$. The classical models of P are $\{a, b\}$, $\{a\}$ and $\emptyset$. The logical consequences of P are the atoms belonging to every model of P. Since $\emptyset \subset \{a\} \subset \{a, b\}$, the set of logical consequences of P is empty, i.e. $\mathrm{Cn}(P) = \emptyset$. Thus we can use the classical models of the program to find its logical consequences. Next consider the intuitive definition of inference rules discussed in Section 2.1. If all the literals in the body of the only rule in program P can be inferred, i.e. if b can be inferred, then we conclude that the head atom a is inferable. The program P itself gives us no means of inferring either a or b. Since we cannot infer b, we also conclude that a is not inferable. Thus in the case of program P the set of logical consequences, which is empty, corresponds to the set of atoms which we intuitively consider inferable. ■

Considering Example 2.7 the set of logical consequences of a positive normal program contains exactly the atoms that are inferable in our intuitive semantics discussed in Section 2.1.

Since the set of logical consequences of a program is a subset of all the models of the program, we define the concept of *minimal model*. A minimal model is a model for which there exists no proper subset that is also a model. Thus a minimal model of a program can be seen as a model that maximizes the falsity of atoms.

Definition 2.8. A model $M \subseteq \mathrm{Hb}(P)$ of a logic program P is minimal if and only if there exists no $M' \subset M$ such that M' is a model of P.

We denote the set of minimal models of a program P by $\mathbf{MM}(P)$. For any positive normal logic program P, it holds that $|\mathbf{MM}(P)| = 1$, i.e. P has a unique minimal model.

Theorem 2.9. For a positive normal logic program P the intersection of all models, $I = \bigcap\{M \subseteq \mathrm{Hb}(P) \mid M \models P\}$, is a model of P. Model I is the unique minimal model, i.e. the *least model* of P.

Proof of Theorem 2.9. Assume that $I \not\models P$. Thus there exists a rule $a \leftarrow b_1, \ldots, b_m \in P$ such that $\{b_1, \ldots, b_m\} \subseteq I$ and $a \notin I$. Consider any $M \subseteq \mathrm{Hb}(P)$ such that $M \models P$. Since $\{b_1, \ldots, b_m\} \subseteq I$ and I is the intersection of all the models of P, it holds that $\{b_1, \ldots, b_m\} \subseteq M$. Since $\{b_1, \ldots, b_m\} \subseteq M$ and $M \models P$, it must hold that $a \in M$. Thus $a \in I = \bigcap\{M \subseteq \mathrm{Hb}(P) \mid M \models P\}$, which is a contradiction. Thus $I \models P$.

Assume that there exists $I' \subset I$ such that $I' \models P$. Since $I = \bigcap\{M \subseteq \mathrm{Hb}(P) \mid M \models P\}$, it holds that $I \subseteq I'$, which is a contradiction. Thus I is a minimal model of P.

Assume that P has two minimal models I_1 and I_2, $I_1 \neq I_2$. Their intersection $I_1 \cap I_2$ is also a model of P. Assume the opposite, $I_1 \cap I_2 \not\models P$. Thus there exist a rule $a \leftarrow b_1, \ldots, b_m \in P$ such that $\{b_1, \ldots, b_m\} \subseteq I_1 \cap I_2$ ($\Rightarrow \{b_1, \ldots, b_m\} \subseteq I_1$ and $\{b_1, \ldots, b_m\} \subseteq I_2$) and $a \notin I_1 \cap I_2$ ($\Rightarrow$ either $a \notin I_1$ or $a \notin I_2$). This is contradictory to $I_1 \models P$ and $I_2 \models P$. Thus $I_1 \cap I_2 \models P$. Furthermore, $I_1 \cap I_2 = I_1$ and $I_1 \cap I_2 = I_2$ since I_1 and I_2 are both minimal. Thus $I_1 = I_2$, which is a contradiction. Since the minimal model of P is unique, it must be the intersection of all the models of P. □

Let us denote the least model of a program P by $\mathrm{LM}(P) = \bigcap\{M \subseteq \mathrm{Hb}(P) \mid M \models P\}$. If P is a positive normal program, the logical consequences of P are exactly the atoms belonging to the least model of P, i.e. $\mathrm{Cn}(P) = \mathrm{LM}(P)$. Thus for positive normal programs our intuition of semantics corresponds to the minimal model semantics [25].

Now let us extend our consideration to positive disjunctive programs. In the case of positive normal programs we concluded that the logical consequences of a program are exactly the atoms belonging to the unique least model. With disjunctions in the heads of the rules the situation becomes more complicated, since a positive disjunctive logic program does not necessarily have a model or a unique least model. A positive disjunctive logic program can have no minimal models, a unique minimal model or several minimal models. However, if integrity constraints are not allowed, a positive disjunctive logic program has at least one model and thus, a minimal model.

Example 2.10. Consider the following three positive logic programs: $P_1 = \{a.\ \bot \leftarrow a\}$, $P_2 = \{a \mid a\}$ and $P_3 = \{a \mid b\}$. Program P_1 has no models nor minimal models. Program P_2 has a unique model $\{a\}$, which is the minimal model of P_2. Program P_3 has three models: $\{a\}$, $\{b\}$ and $\{a, b\}$, of which $\{a\}$ and $\{b\}$ are minimal. ■

Since minimal models are not unique the set of atoms inferable from a positive disjunctive logic program is not necessarily unique. However, if we give up the uniqueness assumption, the minimal model semantics can also be applied to positive disjunctive programs.

Example 2.11. Consider the following positive logic program P.

$$a \mid b.$$
$$c \leftarrow a.$$
$$c \leftarrow b.$$
$$d \leftarrow a, b$$

Intuitively, based on the first rule in P, we can conclude that either a or b is inferable from P, since if neither of them is inferred, then P is not satisfied. Combined with the second and the third rule, we infer c. Finally by the last rule, we should intuitively assume that d is inferable if both a and b can be inferred. However, all the other rules in P are satisfied even if only one of the atoms a and b is inferred. Therefore d is not inferable from P. Thus intuitively we consider the sets $\{a, c\}$ and $\{b, c\}$ as the sets of atoms inferable from P.

Now, let us consider the minimal models of P. Program P has five models $\{a, c\}$, $\{b, c\}$, $\{a, c, d\}$, $\{b, c, d\}$ and $\{a, b, c, d\}$. It has two minimal models, $\{a, c\}$ and $\{b, c\}$. Hence the minimal models are exactly what we intuitively consider as the sets of atoms inferable from the disjunctive program. ■

2.3 STABLE MODEL SEMANTICS

Now we have a suitable semantics for positive disjunctive logic programs. Using positive programs we can express positive information. However, it is essential in knowledge representation to be able to provide negative information, also. Thus we will now discuss how to extend the semantics to the whole class of disjunctive programs, including the programs containing negation not considered yet. The problem is, how to interpret minimality of models when there are negations in the bodies of rules.

Example 2.12. Consider program $P = \{a \leftarrow \sim b\}$. Intuitively we should conclude that a can be inferred only if b is not inferable (i.e. $\sim b$ can be inferred). The program P has three classical models: $\{a\}$, $\{b\}$ and $\{a, b\}$. Two of these, $M_1 = \{a\}$ and $M_2 = \{b\}$, are minimal. The model M_2 is not compatible with our intuition of inference rules. This shows that minimality is not enough to guarantee the intuitive semantics discussed in Section 2.1 in the case of programs containing negation. For a model to be minimal we have to require that an atom is true in the model only if there exists an explicit reason for it to be true. The only rule in P gives us no reason to assume that b is true. Therefore we assume that it has to be false and reject the model M_2. The model M_1, however, satisfies the intuitive semantics we considered in Section 2.1. ■

The solution to the problems with programs containing negation is the *stable model semantics* [9]. The key idea is to pre-evaluate the negations in the program with respect to a given model candidate and then find the minimal models for the pre-evaluated program. The pre-evaluation preserves satisfiability; if M is a model of program P, then M is also a model for the program that is obtained from P by pre-evaluating it with respect to M (see Theorem 2.19). A model candidate is accepted as a stable model if it is a minimal model of the pre-evaluated positive program.

The semantics for *normal* logic programs in terms of stable models was proposed by Gelfond and Lifschitz [9] and was later generalized for disjunctive logic programs independently by Przymusinski [36] and Gelfond and Lifschitz [11].

To obtain stable models for a disjunctive program P, we need first to define the *Gelfond-Lifschitz reduct* of P with respect to a model candidate M.

Definition 2.13. Given a disjunctive logic program P and an interpretation $M \subseteq \mathrm{Hb}(P)$, the Gelfond-Lifschitz reduct P_M is obtained by following two steps.

1. Delete each rule from P that has a negative default literal $\sim a$ in its body such that $a \in M$.
2. Delete all negative default literals in the bodies of the remaining rules.

Thus $P_M = \{A \leftarrow B \mid A \leftarrow B, \sim C \in P \text{ and } M \cap C = \emptyset\}$.

Example 2.14. Recall the program P in Example 2.3. Given an interpretation $M_1 = \{b\}$, we obtain the Gelfond-Lifschitz reduct P_{M_1} presented in Figure 2.1(a). Similarly, for interpretations $M_2 = \{c, d\}$ and $M_3 = \{b, c\}$, we obtain the reducts P_{M_2} and P_{M_3} as presented in Figures 2.1(b) and (c), respectively. ■

$a \mid b \mid c.$	$a \mid b \mid c.$	
$e \leftarrow b, d.$	$\bot \leftarrow a.$	
$a \leftarrow e, d$	$a \leftarrow e, d.$	$a \mid b \mid c.$
	$a \mid d$	$a \leftarrow e, d$
(a)	(b)	(c)

Figure 2.1: The Gelfond-Lifschitz reducts (a) P_{M_1}, (b) P_{M_2} and (c) P_{M_3} from Example 2.14, where $M_1 = \{b\}$, $M_2 = \{c, d\}$ and $M_3 = \{b, c\}$.

By Definition 2.13 the reduct P_M is positive. We define the stable models of P using the minimal models of the reduct P_M.

Definition 2.15. An interpretation $M \subseteq \mathrm{Hb}(P)$ is a stable model of P if and only if M is a minimal model of P_M.

Thus $M \subseteq \mathrm{Hb}(P)$ is a stable model of program P if and only if $M \in \mathbf{MM}(P_M)$. We denote the set of stable models of P by $\mathbf{SM}(P)$. If P is a positive program, then $\mathbf{SM}(P) = \mathbf{MM}(P)$, since $P = P_M$ for any $M \subseteq \mathrm{Hb}(P)$. Thus stable model semantics coincides with minimal model semantics in the case of positive programs.

For a normal logic program P (a special case of disjunctive programs), the reduct P_M is a positive normal program. Thus by Theorem 2.9 P_M has a unique least model $\mathrm{LM}(P_M)$. Furthermore, $M \in \mathbf{SM}(P)$ if and only if $M = \mathrm{LM}(P_M)$ [9].

Example 2.16. The program P presented in Example 2.3 has two stable models: $M_1 = \{b\}$ and $M_2 = \{c, d\}$. The reduct P_{M_1} is presented in Figure 2.1(a) and has three minimal models: $\{a\}$, $\{b\}$ and $\{c\}$. Thus $M_1 \in \mathbf{MM}(P_{M_1})$ and $M_1 \in \mathbf{SM}(P)$. The reduct P_{M_2} presented in Figure 2.1(b) has two minimal models: $\{b, d\}$ and $\{c, d\}$. Thus $M_2 \in \mathbf{MM}(P_{M_2})$ and $M_2 \in \mathbf{SM}(P)$. On the other hand $M_3 = \{b, c\}$ is not a stable model of P. The reduct P_{M_3} presented in Figure 2.1(c) has three minimal models: $\mathbf{MM}(P_{M_3}) = \{\{a\}, \{b\}, \{c\}\}$. Now $M_3 \notin \mathbf{MM}(P_{M_3})$ and thus $M_3 \notin \mathbf{SM}(P)$. ■

There are two conditions which a stable model M of a program P needs to satisfy. First, M must be a model of P_M. The second condition to satisfy is the minimality of M, thus there cannot exist $M' \subset M$ such that $M' \models P_M$. Thus, when $M \notin \mathbf{SM}(P)$ for a particular interpretation $M \subseteq \mathrm{Hb}(P)$, there are two possibilities. Either

- $M \not\models P_M$, or
- $M \models P_M$ and there exists $M' \subset M$ such that $M' \models P_M$.

Example 2.17. Let us consider program P presented in Example 2.3. The interpretation $M_3 = \{b, c\}$ in Example 2.14 satisfies the first condition: $M_3 \models P_{M_3}$. However, it does not satisfy the second. Consider e.g. $\{b\} \in \mathbf{MM}(P_{M_3})$. Now $\{b\} \subset M_3$ and $\{b\} \models P_{M_3}$. Thus $M_3 \notin \mathbf{MM}(P_M)$. An example of an interpretation that does not satisfy the first condition is $M_4 = \{d\}$. The rule $a \mid b \mid c \in P_{M_4}$ is not satisfied in M_4. Thus $M_4 \not\models P_{M_4}$ and furthermore M_4 is not a stable model of P. ■

2.4 PROPERTIES OF STABLE MODELS

In this section we introduce properties of stable models that will be useful later on.

Gelfond and Lifschitz [11] show that an *extended logic program* (a logic program containing classical negation as well as negation as failure) cannot have two stable models of which one is a proper subset of the other. This *anti-chain property* holds for disjunctive logic programs as well.

Theorem 2.18. The stable models of a disjunctive logic program P form an anti-chain; for any $M_1, M_2 \in \mathbf{SM}(P), M_1 \neq M_2$ it holds $M_1 \not\subseteq M_2$ and $M_2 \not\subseteq M_1$.

Proof of Theorem 2.18. Assume that the anti-chain property does not hold, i.e. there exists a disjunctive program P that has two stable models $M_1, M_2 \in \mathbf{SM}(P)$ such that $M_1 \subset M_2$. Now $P_{M_2} \subseteq P_{M_1}$, since

$$\begin{aligned}
P_{M_2} &= \{A \leftarrow B \mid A \leftarrow B, {\sim}C \in P \text{ and } M_2 \cap C = \emptyset\} \\
&= \{A \leftarrow B \mid A \leftarrow B, {\sim}C \in P \text{ and } (M_1 \cup (M_2 \setminus M_1)) \cap C = \emptyset\} \\
&= \{A \leftarrow B \mid A \leftarrow B, {\sim}C \in P \text{ and } (M_1 \cap C) \cup ((M_2 \setminus M_1) \cap C) = \emptyset\} \\
&= \{A \leftarrow B \mid A \leftarrow B, {\sim}C \in P \text{ and } M_1 \cap C = \emptyset\} \\
&\quad \cap \{A \leftarrow B \mid A \leftarrow B, {\sim}C \in P \text{ and } (M_2 \setminus M_1) \cap C = \emptyset\} \\
&= P_{M_1} \cap \{A \leftarrow B \mid A \leftarrow B, {\sim}C \in P \text{ and } (M_2 \setminus M_1) \cap C = \emptyset\}.
\end{aligned}$$

Since M_1 is a stable model of P, it holds that $M_1 \models P_{M_1}$. Since $P_{M_2} \subseteq P_{M_1}$, it also holds that $M_1 \models P_{M_2}$. Finally, since $M_1 \subset M_2$, we have a contradiction with $M_2 \in \mathbf{MM}(P_{M_2})$. □

Janhunen et al. [19] show that for a *partial stable model* M of P it holds that $M \models P$ if and only if $M \models P_M$. This property also holds for (total) stable models of disjunctive programs. Thus the Gelfond-Lifschitz reduct is compatible with the satisfaction relation.

Theorem 2.19. $M \models P$ if and only if $M \models P_M$.

Proof of Theorem 2.19.
($\Rightarrow$) Assume that $M \models P$ and $M \not\models P_M$. Thus there exists a rule $A \leftarrow B \in P_M$ that is not satisfied in M, i.e. $M \models B$ and $M \not\models \bigvee A$. Since $P_M = \{A \leftarrow B \mid A \leftarrow B, \sim C \in P \text{ and } M \cap C = \emptyset\}$, there exists a rule $A \leftarrow B, \sim C \in P$ such that $M \models \sim C$. Thus $M \not\models A \leftarrow B, \sim C$ which is contradictory to $M \models P$.

($\Leftarrow$) Assume that $M \models P_M$ and $M \not\models P$. There exist a rule $A \leftarrow B, \sim C \in P$ that is not satisfied in M, i.e. $M \models B \cup \sim C$ and $M \not\models \bigvee A$. Since $M \models \sim C$, we have $M \cap C = \emptyset$. Since $M \models B$ and $M \not\models \bigvee A$, the reduct P_M contains the rule $A \leftarrow B$ that is not satisfied in M. This is contradictory to $M \models P_M$. □

Next we define the concept of a *supported model* [1]. Any atom that is true in a supported model M of P has to be supported by a rule in P. As stable models are always supported (see Theorem 2.21 below), the characterization of supported models will be useful later on in justifying the reasons for each atom belonging to a stable model.

Definition 2.20. A model $M \subseteq \mathrm{Hb}(P)$ of P ($M \models P$) is a supported model of P if and only if for each $a \in M$ there exists a rule $A \leftarrow B, \sim C \in P$ such that $a \in A$, $M \models B \cup \sim C$, and $M \not\models \bigvee(A \setminus \{a\})$.

We will denote the set of supported models of P by $\mathbf{SuppM}(P)$. A stable model is always supported.

Theorem 2.21. If M is a stable model of P, then M is a supported model of P, i.e. $\mathbf{SM}(P) \subseteq \mathbf{SuppM}(P)$.

Proof of Theorem 2.21. Assume that there exists $M \subseteq \mathrm{Hb}(P)$ such that $M \in \mathbf{SM}(P)$ and $M \notin \mathbf{SuppM}(P)$. Since $M \notin \mathbf{SuppM}(P)$, there exists $a \in M$ such that for each rule $A \leftarrow B, \sim C \in P$: (i) $a \notin A$, (ii) $M \not\models B \cup \sim C$ or (iii) $M \models \bigvee(A \setminus \{a\})$. Let us define $M' = M \setminus \{a\}$ and show that $M' \models P_M$.

(i) Consider rules of the form $A \leftarrow B, \sim C \in P$ for which $a \notin A$ holds. Each rule of the form $A \leftarrow B \in P_M$, corresponding to a rule for which $a \notin A$ holds, is satisfied in M', since $a \notin A$ and $M \models P_M$.

(ii) Consider rules of the form $A \leftarrow B, \sim C \in P$ for which it holds that $M \not\models B \cup \sim C$. Thus $M \not\models B$ or $M \not\models \sim C$. If $M \models \sim C$, then $A \leftarrow B \in P_M$. Thus for each rule $A \leftarrow B \in P_M$, corresponding to a rule for which it holds that $M \not\models B \cup \sim C$, it holds that $M \not\models B$. Since $M' \subset M$, it also holds that $M' \not\models B$. Thus each rule $A \leftarrow B \in P_M$ considered in this item is satisfied in M' regardless of the satisfiability of A.

(iii) Consider a rule of the form $A \leftarrow B, \sim C \in P$ for which $M \models \bigvee(A \setminus \{a\})$ holds. If $M' \not\models \bigvee A$ and $M' \models B$, then $A \leftarrow B \in P_M$ is not satisfied in M'. Since $M \models \bigvee(A \setminus \{a\})$, it holds that $M' \models \bigvee A$. Thus each rule $A \leftarrow B \in P_M$ considered in this item is satisfied in M' regardless of the satisfiability of B.

By (i), (ii) and (iii) we can conclude $M' \models P_M$ for $M' \subset M$, which is in contradiction with $M \in \mathbf{MM}(P_M)$. Thus $\mathbf{SM}(P) \subseteq \mathbf{SuppM}(P)$. □

A supported model is not necessarily a stable model as its definition lacks the minimality criterion essential to stable models.

Example 2.22. Consider the program $P = \{a \leftarrow a\}$. Program P has one stable model, i.e. $\mathbf{SM}(P) = \{\emptyset\}$, and two supported models, i.e. $\mathbf{SuppM}(P) = \{\emptyset, \{a\}\}$. The atom a supports itself and therefore $\{a\}$ is a supported model of P. The model $\{a\}$ is not a stable model since it is excluded by the minimality condition as $\emptyset \subset \{a\}$. Thus $\mathbf{SuppM}(P) \not\subseteq \mathbf{SM}(P)$. ■

3 EQUIVALENCE OF DISJUNCTIVE LOGIC PROGRAMS

In this chapter we formalize the concept of equivalence for disjunctive logic programs. We introduce notions of equivalence in terms of stable models in Section 3.1. We review basic concepts of computational time-complexity in Section 3.2 and discuss the computational complexity of testing the equivalence of disjunctive logic programs in Section 3.3.

3.1 NOTIONS OF EQUIVALENCE

Lifschitz, Pearce and Valverde [22] recently introduced two notions of equivalence for nested logic programs, namely *weak* and *strong equivalence*. Disjunctive logic programs are a special case of nested programs. Thus we can define these notions of equivalence in the disjunctive case, too.

The first notion of equivalence follows naturally from the stable model semantics.

Definition 3.1. Disjunctive logic programs P and Q are (weakly) equivalent if and only if P and Q have exactly the same stable models, i.e. $\mathbf{SM}(P) = \mathbf{SM}(Q)$. This is denoted by $P \equiv Q$.

We generally use the term equivalence synonymously with weak equivalence.

Example 3.2. Consider the programs $P = \{a \mid b\}$, $Q = \{a \leftarrow \sim b.\ b \leftarrow \sim a\}$ and $R = \{a \mid a\}$. P and Q have the same stable models, i.e. $\mathbf{SM}(P) = \{\{a\}, \{b\}\} = \mathbf{SM}(Q)$, but $\mathbf{SM}(R) = \{\{a\}\}$. Thus $P \equiv Q$ and $P \not\equiv R$. ■

The second notion, strong equivalence, is defined in terms of the weak equivalence.

Definition 3.3. Disjunctive logic programs P and Q are strongly equivalent if and only if $P \cup R \equiv Q \cup R$ for all disjunctive logic programs R. This is denoted by $P \equiv_s Q$.

The logic program R in the definition of strong equivalence can be seen as an arbitrary context for P and Q. Thus P and Q are strongly equivalent if and only if they have exactly the same stable models in every context in which they can be placed. Clearly, $P \equiv_s Q$ implies $P \equiv Q$ (by the empty context $R = \emptyset$), but not vice versa.

Example 3.4. Consider programs $P = \{a \leftarrow \sim c\}$ and $Q = \{a \leftarrow \sim b\}$. P and Q have the same stable models, i.e. $\mathbf{SM}(P) = \{\{a\}\} = \mathbf{SM}(Q)$. Thus $P \equiv Q$. However, if we choose program $R = \{b \leftarrow \sim a\}$ as the context in which P and Q are placed, we notice that P and Q are not strongly equivalent, since $\mathbf{SM}(P \cup R) = \{\{a\}\} \neq \{\{a\}, \{b\}\} = \mathbf{SM}(Q \cup R)$. Thus $P \not\equiv_s Q$. ■

In this work we concentrate on the notion of weak equivalence in the case of disjunctive logic programs.

A naive approach to testing the equivalence of logic programs would involve computing the stable models of one program and checking that they

are also stable models of the other program and vice versa. In the worst case (i.e. when the programs are equivalent) one would have to generate and check all the stable models of both programs. If the programs have many stable models, the naive approach quickly becomes infeasible. Therefore a different approach might be useful.

Clearly testing for the equivalence of programs P and Q involves two directions, i.e. one needs to check that all stable models of P are stable models of Q and all stable models of Q are stable models of P. Let us consider only one direction for now. A counter-example for the equivalence of programs P and Q (in one direction) is an interpretation M such that M is a stable model of P but not of Q. Similarly to the discussion in Section 2.3, there are two reasons for M not to be a stable model of Q: either M is not even a model of Q_M (thus $M \not\models Q$ by Theorem 2.19), or $M \models Q_M$ but M is not a minimal model of Q_M. We will distinguish two types of counter-examples, namely counter-examples of type T1 and T2, and present them as pairs.

Definition 3.5. For any disjunctive logic programs P and Q and interpretations $M, M' \subseteq \mathrm{Hb}(P)$,

- $\langle M, M \rangle$ is a counter-example of type T1 if and only if $M \in \mathbf{SM}(P)$ and $M \not\models Q_M$, and
- $\langle M, M' \rangle$ is a counter-example of type T2 if and only if $M \in \mathbf{SM}(P)$, $M \models Q_M$, $M' \subset M$, and $M' \models Q_M$.

We denote the set of counter-examples of the above-mentioned types T1 and T2 by $\mathrm{CE}(P, Q)$. Thus $P \equiv Q$ if and only if $\mathrm{CE}(P, Q) = \emptyset$ and $\mathrm{CE}(Q, P) = \emptyset$.

The above mentioned counter-examples can also be used to show that programs are not strongly equivalent by forming an SE-model [40]. That is, in case of a type T1 counter-example, the pair $\langle M, M \rangle$ is an SE-model of P but not of Q and in case of type T2, $\langle M', M \rangle$ is an SE-model of Q but not of P. Thus $\mathrm{CE}(P, Q) \neq \emptyset$ implies $P \not\equiv_s Q$.

It is also worth noticing that in case of a counter-example $\langle M, M' \rangle$ of type T2, the interpretation M' is not necessarily a minimal model of Q_M. The number of counter-examples of type T2 can be reduced by insisting on minimality of M', but then translations (to be presented in Chapter 4) need then to be revised accordingly.

3.2 COMPUTATIONAL COMPLEXITY

Let us discuss the time-complexity classes and consider languages $L \subseteq \Sigma^*$ over an alphabet Σ. The discussion and the definitions in this section are mainly based on [32]. We assume that the reader is familiar with the concepts of a *deterministic* and a *nondeterministic Turing machine* as models of computation.

Definition 3.6. A deterministic Turing machine M decides a language L if for any string $x \in \Sigma^*$, if $x \in L$, then M halts at the "yes" state on input x and if $x \notin L$, then M halts at the "no" state on input x.

We say that a computation of a Turing machine M is *accepting*, if M halts at the "yes" state and *rejecting*, if M halts at the "no" state.

Definition 3.7. A nondeterministic Turing machine M decides a language L if for any string $x \in \Sigma^*$, $x \in L$, if and only if there exists an accepting computation of M on input x.

The asymmetry between accepting and rejecting in Definition 3.7 is worth noticing. Just *one* accepting nondeterministic computation is enough for x to be accepted, the other computations may result in rejection. The string x is rejected if *every* nondeterministic computation is rejecting, implying that there exists no accepting computation.

Definition 3.8. Let $L \subseteq \Sigma^*$ be a language. The complement of L, denoted by $\overline{L}$, is the set of strings $x \in \Sigma^*$ such that $x \notin L$, i.e. $\overline{L} = \Sigma^* \setminus L$.

The time-complexity class $\mathbf{P}$ contains the languages that are decidable in polynomial time using a deterministic Turing-machine. The class $\mathbf{NP}$ contains the languages that are decidable in polynomial time using a nondeterministic Turing-machine. The class $\mathbf{coNP}$ contains the languages the complements of which belong to $\mathbf{NP}$, i.e. if language $L \in \mathbf{NP}$, then its complement $\overline{L} \in \mathbf{coNP}$.

Next, we consider oracle Turing machines that extend the computational power of an ordinary Turing machine beyond the classes $\mathbf{P}$, $\mathbf{NP}$ and $\mathbf{coNP}$.

Definition 3.9. A C-oracle can decide every problem in the class C in a unit time. A C-oracle Turing machine is a Turing machine that uses C-oracle calls.

Now, we can define new time-complexity classes using oracle Turing machines. The class $\mathbf{P}^C$ contains the languages that are decidable in polynomial time using a deterministic C-oracle Turing machine. The class $\mathbf{NP}^C$ contains the languages that are decidable in polynomial time using a nondeterministic C-oracle Turing-machine and the class $\mathbf{coNP}^C$ contains the languages the complements of which belong to $\mathbf{NP}^C$

The classes $\mathbf{P}$, $\mathbf{NP}$ and $\mathbf{coNP}$ form the first level of the *polynomial hierarchy* [39] that is defined as follows.

Definition 3.10. The polynomial hierarchy is the following sequence of time-complexity classes:

- $\Delta_0^{\mathbf{P}} = \Sigma_0^{\mathbf{P}} = \Pi_0^{\mathbf{P}} = \mathbf{P}$,
- $\Delta_{k+1}^{\mathbf{P}} = \mathbf{P}^{\Sigma_k^{\mathbf{P}}}$ for $k \geq 0$,
- $\Sigma_{k+1}^{\mathbf{P}} = \mathbf{NP}^{\Sigma_k^{\mathbf{P}}}$ for $k \geq 0$, and
- $\Pi_{k+1}^{\mathbf{P}} = \mathbf{coNP}^{\Sigma_k^{\mathbf{P}}}$ for $k \geq 0$.

In particular, $\Sigma_1^{\mathbf{P}} = \mathbf{NP}$, $\Pi_1^{\mathbf{P}} = \mathbf{coNP}$, $\Sigma_2^{\mathbf{P}} = \mathbf{NP}^{\mathbf{NP}}$ and $\Pi_2^{\mathbf{P}} = \mathbf{coNP}^{\mathbf{NP}}$.

The class $\mathbf{P}^{\Sigma_k^{\mathbf{P}}}$ contains the languages that are decidable in polynomial time using a deterministic Turing-machine that uses $\Sigma_k^{\mathbf{P}}$-oracle calls. A $\Sigma_k^{\mathbf{P}}$-oracle can decide every problem in the class $\Sigma_k^{\mathbf{P}}$ in a unit time. The class $\mathbf{NP}^{\Sigma_k^{\mathbf{P}}}$ contains the languages that are decidable in polynomial time using a nondeterministic $\Sigma_k^{\mathbf{P}}$-oracle Turing machine while $\mathbf{coNP}^{\Sigma_k^{\mathbf{P}}}$ contains

the languages the complements of which belong to $\mathbf{NP}^{\Sigma_k^{\mathbf{P}}}$. Thus the class $\mathbf{NP}^{\mathbf{NP}}$ contains the languages that are decidable in polynomial time using a nondeterministic **NP**-oracle Turing-machine.

To be able to compare the computational complexity of different problems, i.e. to define when a problem is as hard as another, we need the concept of a reduction from one language to another.

Definition 3.11. A language $L_1 \subseteq \Sigma^*$ is reducible to language $L_2 \subseteq \Sigma^*$, if there exists a function $R : \Sigma^* \to \Sigma^*$ that is computable by a deterministic Turing machine in polynomial time with the following property: for all inputs x it holds that $x \in L_1$ if and only if $R(x) \in L_2$. Let us denote this by $L_1 \leq_m L_2$. Function R is a *(polynomial-time many-one) reduction* from L_1 to L_2.

Since reducibility is transitive (that is, if $L_1 \leq_m L_2$ and $L_2 \leq_m L_3$, then $L_1 \leq_m L_3$), the languages can be ordered with respect to their difficulty.

Definition 3.12. A language L is *hard* for a complexity class C (C-hard) if every language $L' \in C$ is reducible to L, i.e. $L' \leq_m L$. A language L is *C-complete* if it is C-hard and $L \in C$.

To solve a decision problem using a Turing machine one needs to decide how to represent instances of the problem as strings over some alphabet Σ. An algorithm for the decision problem is a Turing machine that decides the language corresponding to the "yes"-instances of the problem.

3.3 COMPLEXITY OF EQUIVALENCE TESTING

Now we are ready to discuss the computational complexity of equivalence testing in the disjunctive case. We introduce first the languages SM, NotMin, IMPR, IMPL and EQV corresponding to decision problems of our interest.

Definition 3.13. For any *finite* disjunctive logic programs P and Q, and an interpretation $M \subseteq \mathrm{Hb}(P)$,

- $P \in \mathrm{SM} \iff$ there exists an interpretation $M \subseteq \mathrm{Hb}(P)$ such that $M \in \mathbf{SM}(P)$,
- $(P, M) \in \mathrm{NotMin} \iff$ there exists $M' \subset M$ such that $M' \models P$,
- $(P, Q) \in \mathrm{IMPR} \iff \mathbf{SM}(P) \subseteq \mathbf{SM}(Q)$,
- $(P, Q) \in \mathrm{IMPL} \iff (Q, P) \in \mathrm{IMPR}$, and
- $(P, Q) \in \mathrm{EQV} \iff \mathbf{SM}(P) = \mathbf{SM}(Q) \iff P \equiv Q$.

Let us discuss how the languages in Definition 3.13 are located in the polynomial hierarchy. The language SM for disjunctive logic programs is $\Sigma_2^{\mathbf{P}}$-complete [5]. Its complement, $\overline{\mathrm{SM}}$, is thus $\Pi_2^{\mathbf{P}}$-complete. When restricted to normal logic programs, SM is only **NP**-complete [27] and consequently $\overline{\mathrm{SM}}$ is **coNP**-complete.

Theorem 3.14. Language NotMin is in **NP**.

Proof of Theorem 3.14. We consider the language

$$\text{NotMin} = \{(P, M) \mid \text{there exists } M' \subset M \text{ such that } M' \models P\}.$$

Let us show that NotMin $\in \mathbf{NP}$, i.e. there exists a nondeterministic Turing-machine deciding the language NotMin. Instead of presenting the actual Turing machine we present an abstract algorithm for deciding NotMin that could be used to construct the actual Turing machine. The algorithm is presented in Figure 3.1.

TestNotMinimal(P, M)
1 Select $M' \subset M$
2 **if** $M' \models P$
3 **then return** "yes"
4 **else return** "no"
5 **fi**

Figure 3.1: The algorithm for deciding NotMin.

Let us discuss the algorithm. First, since we are using a nondeterministic computation, we can choose an interpretation M' in polynomial time. Also, the test on line 2 can be done in polynomial time (one has to check whether each rule in P is satisfied in M'). Now, **TestNotMinimal**(P, M) has a accepting computation on $(P, M) \iff (P, M) \in$ NotMin. Thus we can construct a nondeterministic Turing machine deciding NotMin and therefore NotMin $\in \mathbf{NP}$. □

By Theorem 3.14, an **NP**-oracle can be used to decide if there exists a model M' of P such that $M' \subset M$ for a given interpretation $M \subseteq \text{Hb}(P)$.

Next, we consider the computational complexity of testing the equivalence of disjunctive logic programs. Lin [24] and Turner [40] have independently shown that testing the *strong equivalence* of disjunctive logic programs is **coNP**-complete. The strong equivalence can be characterized as equivalence in Heyting's logic of *here-and-there* (HT) [22]. As shown by Pearce, Tompits and Woltran [34], satisfiability in here-and-there is reducible to propositional satisfiability (SAT). This implies that existing SAT solvers can be used for testing the strong equivalence of logic programs. Recently, Lin [24] reduced strong equivalence directly to propositional entailment without using HT as an intermediate logic.

Turner [40] also shows that testing the weak equivalence is $\Pi_2^{\mathbf{P}}$-hard by reducing $\overline{\text{SM}}$ to EQV. We show that testing the weak equivalence of disjunctive logic programs is indeed $\Pi_2^{\mathbf{P}}$-complete and use a similar method as Turner in the hardness part of the proof.

Theorem 3.15. IMPR is $\Pi_2^{\mathbf{P}}$-complete.

Proof of Theorem 3.15. To show that IMPR is $\Pi_2^{\mathbf{P}}$-complete we need to show that (i) IMPR $\in \Pi_2^{\mathbf{P}}$ and (ii) a $\Pi_2^{\mathbf{P}}$-complete language (such as $\overline{\text{SM}}$) can be reduced in polynomial time to IMPR, i.e. IMPR is $\Pi_2^{\mathbf{P}}$-hard.

(i) Let us consider the complement of IMPR,

$$\overline{\text{IMPR}} = \{(P, Q) \mid \mathbf{SM}(P) \not\subseteq \mathbf{SM}(Q)\}.$$

Thus $(P, Q) \in \overline{\text{IMPR}} \iff$ there exists $M \in \mathbf{SM}(P)$ such that $M \notin \mathbf{SM}(Q)$. Let us show that $\overline{\text{IMPR}} \in \Sigma_2^{\mathbf{P}}$, i.e. there exists a nondeterministic **NP**-oracle Turing-machine, that decides the language $\overline{\text{IMPR}}$. The abstract algorithm for deciding $\overline{\text{IMPR}}$ is presented in Figure 3.2.

Let us discuss the algorithm. Using nondeterminism, we can choose an interpretation M. Checking whether $M \models P$ on line 2 can be performed in polynomial time checking whether each rule in P is satisfied in M). If $P \models M$, then $M \models P_M$ by Theorem 2.19. The reduct P_M can be constructed in polynomial time. Since $M \models P_M$, we use an **NP**-oracle to check whether $M \in \mathbf{MM}(P_M)$ on line 5. If $M \in \mathbf{MM}(P_M)$, then $M \in \mathbf{SM}(P)$. The test on line 8 can be performed in polynomial time. If $M \not\models Q$, the pair $\langle M, M \rangle$ is a counter-example of type T1. Otherwise, $M \models Q_M$ by Theorem 2.19, and we use an **NP**-oracle (line 11) to test whether $M \in \mathbf{MM}(Q_M)$. If $M \notin \mathbf{MM}(Q_M)$, there exists a counter-example of type T2.

Thus **TestNotInIMPR**(P, Q) has an accepting computation on (P, Q) $\iff$ $(P, Q) \in \overline{\text{IMPR}}$. Therefore $\overline{\text{IMPR}} \in \Sigma_2^{\mathbf{P}}$ and furthermore $\text{IMPR} \in \Pi_2^{\mathbf{P}}$.

(ii) Consider an arbitrary disjunctive logic program R. Now $R \in \overline{\text{SM}} \iff R \notin \text{SM} \iff \mathbf{SM}(R) = \emptyset$. Let $\{\bot\}$ be a program having no stable models, i.e. $\mathbf{SM}(\{\bot\}) = \emptyset$. Thus $\mathbf{SM}(R) \subseteq \mathbf{SM}(\{\bot\}) \iff (R, \{\bot\}) \in \text{IMPR}$. Therefore $R \in \overline{\text{SM}} \iff (R, \{\bot\}) \in \text{IMPR}$, and the reduction is polynomial.

(i) and (ii) imply that IMPR is $\Pi_2^{\mathbf{P}}$-complete. □

TestNotInIMPR(P, Q)
1 Select M
2 **if** $M \not\models P$
3 **then return** "no"
4 **fi**
5 **if** **TestNotMinimal**$(P_M, M) =$ "yes"
6 **then return** "no"
7 **fi**
8 **if** $M \not\models Q$
9 **then return** "yes"
10 **else**
11 **if** **TestNotMinimal**$(Q_M, M) =$ "yes"
12 **then return** "yes"
13 **else return** "no"
14 **fi**
15 **fi**

Figure 3.2: The algorithm for deciding $\overline{\text{IMPR}}$.

Theorem 3.16. IMPL is $\Pi_2^{\mathbf{P}}$-complete.

Proof of Theorem 3.16. IMPL and IMPR can be reduced to each other, i.e.

$$\text{IMPL} =_m \text{IMPR}.$$

The proof divides into two parts: (i) IMPL $\leq_m$ IMPR and (ii) IMPR $\leq_m$ IMPL. Let $A = \{(P, Q) \mid P \text{ and } Q \text{ are disjunctive logic programs}\}$.

(i) Let $R_1 : A \to A$ such that $R_1((P, Q)) = (Q, P)$. By Definition 3.13, $(P, Q) \in \text{IMPL} \iff (Q, P) = R_1((P, Q)) \in \text{IMPR}$. Thus R_1 is a reduction from IMPL to IMPR, and IMPL $\leq_m$ IMPR.

(ii) Let $R_2 : A \to A$ such that $R_2((P, Q)) = (Q, P)$. By Definition 3.13, $(P, Q) \in \text{IMPR} \iff (Q, P) = R_2((P, Q)) \in \text{IMPL}$. Thus R_2 is a reduction from IMPR to IMPL, and IMPR $\leq_m$ IMPL.

Since $\Pi_2^{\mathbf{P}}$ is closed under reductions (since **coNP** is closed under reductions) and IMPR is $\Pi_2^{\mathbf{P}}$-complete, it holds that IMPL is $\Pi_2^{\mathbf{P}}$-complete. □

Theorem 3.17. EQV is $\Pi_2^{\mathbf{P}}$-complete.

Proof of Theorem 3.17. To show that EQV is $\Pi_2^{\mathbf{P}}$-complete we need to show that (i) EQV $\in \Pi_2^{\mathbf{P}}$ and (ii) a $\Pi_2^{\mathbf{P}}$-complete language ($\overline{\text{SM}}$) can be reduced in polynomial time to EQV, i.e. EQV is $\Pi_2^{\mathbf{P}}$-hard.

(i) Let P and Q be arbitrary disjunctive logic programs. Now $(P, Q) \in \text{EQV} \iff \mathbf{SM}(P) = \mathbf{SM}(Q) \iff \mathbf{SM}(P) \subseteq \mathbf{SM}(Q)$ and $\mathbf{SM}(Q) \subseteq \mathbf{SM}(P) \iff (P, Q) \in \text{IMPR}$ and $(P, Q) \in \text{IMPL}$. Thus $\text{EQV} = \text{IMPR} \cap \text{IMPL}$. Since $\text{IMPR} \in \Pi_2^{\mathbf{P}}$, $\text{IMPL} \in \Pi_2^{\mathbf{P}}$ and $\Pi_2^{\mathbf{P}}$ is closed under intersection (since **coNP** is closed under intersection), we have $\text{EQV} \in \Pi_2^{\mathbf{P}}$.

(ii) Let R be an arbitrary disjunctive logic program. Now $R \in \overline{\text{SM}} \iff R \notin \text{SM} \iff \mathbf{SM}(R) = \emptyset$. Let $\{\bot\}$ be a program having no stable models, i.e. $\mathbf{SM}(\{\bot\}) = \emptyset$. Thus $\mathbf{SM}(R) = \emptyset \iff \mathbf{SM}(R) = \mathbf{SM}(\{\bot\}) \iff (R, \{\bot\}) \in \text{EQV}$. Therefore $R \in \overline{\text{SM}} \iff (R, \{\bot\}) \in \text{EQV}$, and the reduction is polynomial.

Thus (i) and (ii) imply that EQV is $\Pi_2^{\mathbf{P}}$-complete. □

Theorem 3.17 implies that testing the weak equivalence of disjunctive logic programs is as hard as deciding whether a disjunctive logic program has stable models or not. Thus there is no complexity theoretical obstacle for developing a polynomial transformation for testing the equivalence of disjunctive logic programs as introduced in [20] for *weight constraint programs*. However, a polynomial many-to-one reduction preserves only the yes/no answers of the decision problem. The reduction itself can be arbitrary as long as it is computable in polynomial time using a deterministic Turing machine. The question remains whether it is possible or not to find a useful systematic method to obtain the transformation similar to the one in [20].

Furthermore, one might think that since deciding $\equiv_s$ for finite propositional disjunctive programs is only **coNP**-complete [34, 24] and $P \equiv_s Q$ implies $P \equiv Q$, there is no need to consider the computationally harder task of deciding $P \equiv Q$. However, the question whether $P \equiv Q$ holds remains open whenever $P \not\equiv_{\text{s}} Q$ turns out to be the case. This implies that verifying $P \equiv Q$ remains as a problem of its own, which cannot be fully compensated by verifying $P \equiv_{\text{s}} Q$.

4 TRANSLATIONS FOR EQUIVALENCE TESTING

In this chapter we present a translation-based method for testing the equivalence of disjunctive logic programs. The idea is to transform disjunctive programs P and Q into a program, the stable models of which can be used to determine the equivalence. We present two alternative translations — one-phased $\mathrm{TR}(P,Q)$ in Section 4.1 and two-phased $[\mathrm{TR}_1(P,Q), \mathrm{TR}_2(P,Q)]$ in Section 4.2. The translation $\mathrm{TR}(P,Q)$ captures both types of counter-examples for the equivalence if such exist. The second translation is employed in two phases. First, translation $\mathrm{TR}_1(P,Q)$ is used to capture counter-examples of type T1. If there are no counter-examples of type T1, then translation $\mathrm{TR}_2(P,Q)$ is used to capture counter-examples of type T2.

4.1 ONE-PHASED TRANSLATION

The idea behind the translation presented in this section is to transform two disjunctive logic programs P and Q into one logic program $\mathrm{TR}(P,Q)$ that has a stable model if and only if P has a stable model that is not a stable model of Q. There are two possible reasons for M not being a stable model of Q corresponding to counter-examples of type T1 and T2: either M is not even a model of Q_M (T1), or M is a model of Q_M but not a minimal model of Q_M (T2). The rules of the translation are used to capture these two possibilities. A further objective is that we should be able to construct counter-examples for the equivalence using the stable models of the translation $\mathrm{TR}(P,Q)$. Thus it should be easy to extract a stable model of P that is not a stable model of Q from a stable model of the translation $\mathrm{TR}(P,Q)$.

We assume that $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$. This is not a significant restriction, since any logic program can be extended with rules of the form $a \leftarrow a$ without affecting the stable models of the program, i.e. $\{a \leftarrow a\} \equiv_s \emptyset$. In the translations to be presented we will also need several new atoms not appearing in P or Q. We introduce for each atom $a \in \mathrm{Hb}(P)$ new atoms $a^\bullet$ and a°. We define $A^\bullet = \{a^\bullet \mid a \in A\}$ and $A^\circ = \{a^\circ \mid a \in A\}$ for any set of atoms A.

The Herbrand base of $\mathrm{TR}(P,Q)$ is to contain new atoms diff, unsat, $\mathit{unsat}^\bullet$ and ok, all the atoms in $\mathrm{Hb}(P)$, and in addition two renamed copies of each atom in $\mathrm{Hb}(P)$. Thus

$$\mathrm{Hb}(\mathrm{TR}(P,Q)) = \{\mathit{diff}, \mathit{unsat}, \mathit{unsat}^\bullet, \mathit{ok}\} \cup \mathrm{Hb}(P) \cup \mathrm{Hb}(P)^\bullet \cup \mathrm{Hb}(P)^\circ.$$

Let us discuss the intended meaning of the new atoms introduced in the translation $\mathrm{TR}(P,Q)$ before discussing the details of the translation. Given $M \in \mathbf{SM}(P)$, the atoms in $\mathrm{Hb}(P)^\bullet \cup \mathrm{Hb}(P)^\circ$ are used in selecting a sub-model M' of M such that $M' \subset M$. The intended meaning of the new atoms are as follows.

unsat – indicates that $M \not\models Q_M$.

$a^\bullet$ – atom $a \in M$ is true in the sub-model M' searched for Q_M

a° – atom $a \in M$ is false in the sub-model $M' \subseteq M$.

$unsat^\bullet$ – indicates that $M' \not\models Q_M$.

diff – indicates that $M' \subseteq \mathrm{Hb}(P)$ is a proper subset of $M \subseteq \mathrm{Hb}(P)$, i.e. $M' \neq M$.

ok – indicates that a counter-example for the equivalence is found.

Now we are ready to introduce the actual translation.

Definition 4.1. Let P and Q be disjunctive logic programs such that $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$. Let $\mathit{diff}, unsat, unsat^\bullet$ and ok be atoms appearing in neither P nor Q. The translation $\mathrm{TR}(P, Q)$ contains the following rules:

1. all the rules of P without modifications,
2. a rule $unsat \leftarrow B, {\sim}(A \cup C)$ for each rule $A \leftarrow B, {\sim}C \in Q$,
3. rules $a^\bullet \leftarrow a, {\sim}a^\circ, {\sim}unsat$ and $a^\circ \leftarrow a, {\sim}a^\bullet, {\sim}unsat$ for each atom $a \in \mathrm{Hb}(P)$,
4. a rule $unsat^\bullet \leftarrow B^\bullet, {\sim}(A^\bullet \cup C), {\sim}unsat$ for each rule $A \leftarrow B, {\sim}C \in Q$,
5. a rule $\mathit{diff} \leftarrow a, {\sim}a^\bullet, {\sim}unsat$ for each atom $a \in \mathrm{Hb}(P)$, and
6. rules $ok \leftarrow unsat$, $ok \leftarrow \mathit{diff}, {\sim}unsat, {\sim}unsat^\bullet$ and $\bot \leftarrow {\sim}ok$.

Let us discuss the meaning of each part of the translation in detail. The items below correspond respectively to the items in Definition 4.1. For an interpretation N of the translation $\mathrm{TR}(P, Q)$ we define $M = N \cap \mathrm{Hb}(P)$.

1. Capture a stable model M of P.
2. Check whether $M \models Q_M$. If there exists a rule in Q_M that is not satisfied in M, $unsat$ is implied.
3. If $M \models Q_M$ and $unsat$ is not implied, the renamed atoms are used in order to select an interpretation M' such that $M' \subseteq M$. These rules force that if $a \in M$, then either a is true in the sub-model candidate M' ($a^\bullet \in N$) or not ($a^\circ \in N$). As seen in Example 3.2, this selection can also be expressed equivalently using a rule $a^\bullet \mid a^\circ \leftarrow a, {\sim}unsat$ for each $a \in \mathrm{Hb}(P)$. We decided, however, to keep the number of disjunctions as low as possible.
4. Check whether $(M')^\bullet \models (Q_M)^\bullet \iff M' \models Q_M$, where $(M')^\bullet = N \cap \mathrm{Hb}(P)^\bullet$. If there exists a rule in Q_M that is not satisfied in M', $unsat^\bullet$ is implied.
5. Check that M' is a proper subset of M. If $M' \subset M$, then *diff* is implied.
6. Summarize the reasons for M not being a stable model of Q. Either
   - T1: $M \not\models Q_M$, and therefore $unsat \in N$, or
   - T2: $M \models Q_M$ and $M \notin \mathbf{MM}(Q_M)$. Thus $unsat \notin N$ (since $M \models Q_M$), $unsat^\bullet \notin N$ (since $M' \models Q_M$), and $\mathit{diff} \in N$ (since $M' \subset M$).

A stable model for the translation exists if and only if such a reason exists. The integrity constraint ensures that every stable model of the translation contains the atom ok.

A counter-example can easily be extracted from a stable model N of translation $\mathrm{TR}(P,Q)$. If $unsat \in N$, we know that $M = N \cap \mathrm{Hb}(P) \in \mathbf{SM}(P)$ and $M \not\models Q_M$. Thus $\langle M, M\rangle$ is a counter-example of type T1. On the other hand, if $unsat \notin N$, we find the counter-example similarly: $M = N \cap \mathrm{Hb}(P) \in \mathbf{SM}(P)$, $M \models Q_M$ and $M \notin \mathbf{MM}(Q_M)$. Furthermore, $M' = \{a \mid a^\bullet \in N \cap \mathrm{Hb}(P)^\bullet\} \subset M$ and $M' \models Q_M$. Thus $\langle M, M'\rangle$ is a counter-example of type T2.

$$
\begin{array}{l}
a \mid b. \\
unsat \leftarrow {\sim}a, {\sim}b. \\
a^\bullet \leftarrow a, {\sim}a^\circ, {\sim}unsat. \\
a^\circ \leftarrow a, {\sim}a^\bullet, {\sim}unsat. \\
b^\bullet \leftarrow b, {\sim}b^\circ, {\sim}unsat. \\
b^\circ \leftarrow b, {\sim}b^\bullet, {\sim}unsat. \\
unsat^\bullet \leftarrow {\sim}a^\bullet, {\sim}b, {\sim}unsat. \\
\mathit{diff} \leftarrow a, {\sim}a^\bullet, {\sim}unsat. \\
\mathit{diff} \leftarrow b, {\sim}b^\bullet, {\sim}unsat. \\
ok \leftarrow unsat. \\
ok \leftarrow \mathit{diff}, {\sim}unsat, {\sim}unsat^\bullet. \\
\bot \leftarrow {\sim}ok
\end{array}
$$

(a)

$$
\begin{array}{l}
a \mid b. \\
a^\bullet \leftarrow a. \\
a^\circ \leftarrow a. \\
b^\circ \leftarrow b. \\
\mathit{diff} \leftarrow a. \\
\mathit{diff} \leftarrow b. \\
ok \leftarrow unsat. \\
ok \leftarrow \mathit{diff}
\end{array}
$$

(b)

Figure 4.1: (a) The translation $\mathrm{TR}(P,Q)$ from Example 4.2 and (b) the reduct $\mathrm{TR}(P,Q)_N$ for interpretation $N = \{b, b^\circ, \mathit{diff}, ok\}$.

Example 4.2. Let us consider the programs $P = \{a \mid b\}$ and $Q = \{a \leftarrow {\sim}b\}$. Program P has two stable models, $\mathbf{SM}(P) = \{\{a\}, \{b\}\}$, while program Q has one, $\mathbf{SM}(Q) = \{\{a\}\}$. The translation $\mathrm{TR}(P,Q)$ is presented in Figure 4.1(a). Let us consider a model candidate $N = \{b, b^\circ, \mathit{diff}, ok\}$. The reduct $\mathrm{TR}(P,Q)_N$ is presented in Figure 4.1(b). We have

$$\mathbf{MM}(\mathrm{TR}(P,Q)_N) = \{\{a, a^\bullet, a^\circ, \mathit{diff}, ok\}, \{b, b^\circ, \mathit{diff}, ok\}\}.$$

Thus $N \in \mathbf{SM}(\mathrm{TR}(P,Q))$. Since the translation has a stable model, we can conclude, that $P \not\equiv Q$, and $M = N \cap \mathrm{Hb}(P) = \{b\}$ can be used to construct a counter-example for the equivalence. Now $M \in \mathbf{SM}(P)$, $M \models Q_M$ and $M \notin \mathbf{MM}(Q_M) = \{\emptyset\}$. We have $M' = \{a \mid a^\bullet \in N \cap \mathrm{Hb}(P)^\bullet\} = \emptyset \subset M$ and $M' \models Q_M$, since $Q_M = \emptyset$. Since $unsat \notin N$, the pair $\langle M, M'\rangle$ is a counter-example of type T2. As both stable models of P are models of Q, there exists no counter-example of type T1. This can also be verified from the translation, since the first two rules in $\mathrm{TR}(P,Q)$ ensure that $unsat$ cannot belong to a stable model of $\mathrm{TR}(P,Q)$. ■

4.1.1 Correctness of the Translation $\mathrm{TR}(P, Q)$

In this section we establish the correctness of the translation $\mathrm{TR}(P, Q)$, showing that $\mathrm{TR}(P, Q)$ has a stable model if and only if P has a stable model that is not a stable model of Q. As a matter a fact we can prove an even stronger result as there is a tight one-to-one correspondence between the sets $\mathrm{CE}(P, Q)$ and $\mathbf{SM}(\mathrm{TR}(P, Q))$. To show this correspondence, we first define mappings from $2^{\mathrm{Hb}(P)} \times 2^{\mathrm{Hb}(Q)}$ to $2^{\mathrm{Hb}(\mathrm{TR}(P,Q))}$ and vice versa.

Definition 4.3. Given two disjunctive logic programs P and Q such that $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ the function

$$\mathrm{EXT}_{P,Q} : 2^{\mathrm{Hb}(P)} \times 2^{\mathrm{Hb}(Q)} \rightarrow 2^{\mathrm{Hb}(\mathrm{TR}(P,Q))}$$

is defined as follows:

$$\begin{aligned}&\mathrm{EXT}_{P,Q}(M, M') \\ &= \begin{cases} M \cup \{unsat, ok\}, \text{ if } M = M', \\ M \cup \{\mathit{diff}, ok\} \cup \{a^\bullet \mid a \in M'\} \cup \{a^\circ \mid a \in M \setminus M'\}, \text{otherwise.} \end{cases}\end{aligned}$$

Definition 4.4. Given the translation $\mathrm{TR}(P, Q)$ of disjunctive logic programs P and Q, the function

$$\mathrm{PROJ}_{P,Q} : 2^{\mathrm{Hb}(\mathrm{TR}(P,Q))} \rightarrow 2^{\mathrm{Hb}(P)} \times 2^{\mathrm{Hb}(Q)}$$

is defined as follows:

$$\mathrm{PROJ}_{P,Q}(N) = \begin{cases} \langle N \cap \mathrm{Hb}(P), N \cap \mathrm{Hb}(P)\rangle, \text{ if } unsat \in N, \\ \langle N \cap \mathrm{Hb}(P), \{a \in \mathrm{Hb}(P) \mid a^\bullet \in N\}\rangle, \text{otherwise.} \end{cases}$$

The form of the reduct $\mathrm{TR}(P, Q)_N$ for a given interpretation N is as follows.

Lemma 4.5. Given an interpretation N for $\mathrm{TR}(P, Q)$ from Definition 4.1, let us define $M_1 = N \cap \mathrm{Hb}(P)$, $M_2 = \{a \in \mathrm{Hb}(P) \mid a^\bullet \in N\}$ and $R = \{a \in \mathrm{Hb}(P) \mid a^\circ \in N\}$. Thus ${M_2}^\bullet = N \cap \mathrm{Hb}(P)^\bullet$ and $R^\circ = N \cap \mathrm{Hb}(P)^\circ$. The reduct $\mathrm{TR}(P, Q)_N$ contains the following rules:

1. all the rules of P_{M_1},
2. a rule $unsat \leftarrow B \iff$ there exists a rule $A \leftarrow B, {\sim}C \in Q$ such that $M_1 \not\models \bigvee A$ and $M_1 \models {\sim}C$,
3. the following $\iff unsat \notin N$:
   (a) a rule $a^\bullet \leftarrow a \iff$ there exists $a \in \mathrm{Hb}(P)$ such that $a \notin R$,
   (b) a rule $a^\circ \leftarrow a \iff$ there exists $a \in \mathrm{Hb}(P)$ such that $a \notin M_2$,
   (c) a rule $unsat^\bullet \leftarrow B^\bullet \iff$ there exists a rule $A \leftarrow B, {\sim}C \in Q$ such that $M_2 \not\models \bigvee A$ and $M_1 \models {\sim}C$,
   (d) a rule $\mathit{diff} \leftarrow a \iff$ there exists $a \in \mathrm{Hb}(P)$ such that $a \notin M_2$,

4. the rule $ok \leftarrow unsat$,

5. the rule $ok \leftarrow \mathit{diff} \iff unsat \notin N$ and $unsat^\bullet \notin N$, and

6. the rule $\bot \iff ok \notin N$.

Proof of Lemma 4.5. We examine the rules in each item of Lemma 4.5 separately, and use Definitions 4.1 and 2.13.

1. $A \leftarrow B \in \mathrm{TR}(P,Q)_N$
   $\iff A \leftarrow B, {\sim}C \in \mathrm{TR}(P,Q)$ and $C \cap N = \emptyset$
   $\iff A \leftarrow B, {\sim}C \in P$ and $C \cap M_1 = \emptyset$
   $\iff A \leftarrow B \in P_{M_1}$.

2. $unsat \leftarrow B \in \mathrm{TR}(P,Q)_N$
   $\iff unsat \leftarrow B, {\sim}(A \cup C) \in \mathrm{TR}(P,Q)$ and $(A \cup C) \cap N = \emptyset$
   $\iff unsat \leftarrow B, {\sim}(A \cup C) \in \mathrm{TR}(P,Q)$ and $(A \cup C) \cap M_1 = \emptyset$
   $\iff unsat \leftarrow B, {\sim}(A \cup C) \in \mathrm{TR}(P,Q)$, $M_1 \not\models \bigvee A$ and $M_1 \models {\sim}C$
   $\iff A \leftarrow B, {\sim}C \in Q$, $M_1 \not\models \bigvee A$ and $M_1 \models {\sim}C$.

3. If $unsat \notin N$ and $a \in \mathrm{Hb}(P)$:

   (a) $a^\bullet \leftarrow a \in \mathrm{TR}(P,Q)_N$
   $\iff a^\bullet \leftarrow a, {\sim}a^\circ, {\sim}unsat \in \mathrm{TR}(P,Q)$ and $a^\circ \notin N$
   $\iff a^\bullet \leftarrow a, {\sim}a^\circ, {\sim}unsat \in \mathrm{TR}(P,Q)$ and $a^\circ \notin R^\circ$
   $\iff a^\bullet \leftarrow a, {\sim}a^\circ, {\sim}unsat \in \mathrm{TR}(P,Q)$ and $a \notin R$.

   (b) $a^\circ \leftarrow a \in \mathrm{TR}(P,Q)_N$
   $\iff a^\circ \leftarrow a, {\sim}a^\bullet, {\sim}unsat \in \mathrm{TR}(P,Q)$ and $a^\bullet \notin N$
   $\iff a^\circ \leftarrow a, {\sim}a^\bullet, {\sim}unsat \in \mathrm{TR}(P,Q)$ and $a^\bullet \notin M_2{}^\bullet$
   $\iff a^\circ \leftarrow a, {\sim}a^\bullet, {\sim}unsat \in \mathrm{TR}(P,Q)$ and $a \notin M_2$.

   (c) $unsat^\bullet \leftarrow B^\bullet \in \mathrm{TR}(P,Q)_N$
   $\iff unsat^\bullet \leftarrow B^\bullet, {\sim}(A^\bullet \cup C), {\sim}unsat \in \mathrm{TR}(P,Q)$ such that $(A^\bullet \cup C) \cap N = \emptyset$
   $\iff unsat^\bullet \leftarrow B^\bullet, {\sim}(A^\bullet \cup C), {\sim}unsat \in \mathrm{TR}(P,Q)$ such that $A^\bullet \cap M_2{}^\bullet = \emptyset$ and $C \cap M_1 = \emptyset$
   $\iff A \leftarrow B, {\sim}C \in Q$, $M_2 \not\models \bigvee A$ and $M_1 \models {\sim}C$.

   (d) $\mathit{diff} \leftarrow a \in \mathrm{TR}(P,Q)_N$
   $\iff \mathit{diff} \leftarrow a, {\sim}a^\bullet, {\sim}unsat \in \mathrm{TR}(P,Q)$ and $a^\bullet \notin N$
   $\iff \mathit{diff} \leftarrow a, {\sim}a^\bullet, {\sim}unsat \in \mathrm{TR}(P,Q)$ and $a^\bullet \notin M_2{}^\bullet$
   $\iff \mathit{diff} \leftarrow a, {\sim}a^\bullet, {\sim}unsat \in \mathrm{TR}(P,Q)$ and $a \notin M_2$.

4. $ok \leftarrow unsat \in \mathrm{TR}(P,Q)_N \iff ok \leftarrow unsat \in \mathrm{TR}(P,Q)$.

5. $ok \leftarrow \mathit{diff} \in \mathrm{TR}(P,Q)_N$
   $\iff ok \leftarrow \mathit{diff}, {\sim}unsat, {\sim}unsat^\bullet \in \mathrm{TR}(P,Q)$, $unsat \notin N$ and $unsat^\bullet \notin N$.

6. $\bot \in \mathrm{TR}(P,Q)_N \iff \bot \leftarrow {\sim}ok \in \mathrm{TR}(P,Q)$ and $ok \notin N$. □

The extension N of a counter-example for the equivalence of programs P and Q is a model of the reduct $\mathrm{TR}(P,Q)_N$.

Lemma 4.6. If the pair $\langle M, M' \rangle \in \mathrm{CE}(P, Q)$, then $N \models \mathrm{TR}(P, Q)_N$ for $N = \mathrm{EXT}_{P,Q}(M, M')$.

Proof of Lemma 4.6. Assume $\langle M, M' \rangle \in \mathrm{CE}(P, Q)$. First, it is worth noticing that by Definition 4.3 and since $M' \subseteq M$, if $a^\bullet \in N$ for an atom $a \in \mathrm{Hb}(P)$, then $a^\circ \notin N$ and vice versa. The pairs in $\mathrm{CE}(P, Q)$ divide in two cases corresponding to counter-examples of types (i) T1 and (ii) T2.

(i) Assume $\langle M, M' \rangle \in \mathrm{CE}(P, Q)$ is of type T1. Then $M = M'$, $M \in \mathbf{SM}(P)$ and $M \not\models Q_M$. Then by Definition 4.3, it holds $N = M \cup \{unsat, ok\}$. Thus, by Lemma 4.5, we have

$$\begin{aligned} \mathrm{TR}(P, Q)_N \;=\; & P_M \\ & \cup \{unsat \leftarrow B \mid A \leftarrow B, {\sim}C \in Q \text{ and } M \models {\sim}(A \cup C)\} \\ & \cup \{ok \leftarrow unsat\}. \end{aligned}$$

Now, since $M \in \mathbf{SM}(P)$, $M \models P_M$ and furthermore $N \models P_M$ as $M = N \cap \mathrm{Hb}(P)$. Since $unsat \in N$, each rule of the form $unsat \leftarrow B$ is satisfied regardless of the satisfiability of B. Furthermore, the rule $ok \leftarrow unsat$ is satisfied in N. Thus $N \models \mathrm{TR}(P, Q)_N$.

(ii) Assume $\langle M, M' \rangle \in \mathrm{CE}(P, Q)$ is of type T2. Thus $M \in \mathbf{SM}(P)$, $M \models Q_M$, $M' \subset M$ and $M' \models Q_M$. Then by Definition 4.3,

$$N = M \cup \{\mathit{diff}, ok\} \cup \{a^\bullet \mid a \in M'\} \cup \{a^\circ \mid a \in M \setminus M'\}.$$

Let us show that each rule in the reduct $\mathrm{TR}(P, Q)_N$ is satisfied, the items below corresponding to the items in Lemma 4.5, where $M_1 = M$, $M_2 = M'$ and $R = M \setminus M'$.

1. Since $M \in \mathbf{SM}(P)$, $M \models P_M$, and thus $N \models P_M$.
2. By Lemma 4.5 a rule $unsat \leftarrow B \in \mathrm{TR}(P, Q)_N$, if there exists a rule $A \leftarrow B, {\sim}C \in Q$ such that $M \not\models \bigvee A$ and $M \models {\sim}C$. Since $M \models Q_M$, $M \models Q$ by Theorem 2.19. Thus for each rule $A \leftarrow B, {\sim}C \in Q$, if $M \not\models \bigvee A$ and $M \models {\sim}C$, then $M \not\models B$. If $M \not\models B$, then, since $M = N \cap \mathrm{Hb}(P)$ and $B \subseteq \mathrm{Hb}(P)$, $N \not\models B$. Thus each rule of the form $unsat \leftarrow B \in \mathrm{TR}(P, Q)_N$ is satisfied in N.

3a. By Lemma 4.5 a rule $a^\bullet \leftarrow a \in \mathrm{TR}(P, Q)_N$ ($a \in \mathrm{Hb}(P)$), if $a \notin M \setminus M'$. If $a \notin N$, the rule is satisfied in N regardless of the satisfiability of $a^\bullet$. If $a \in N$, then $a^\bullet \in N$ (since, if $a \notin M \setminus M'$, then $a^\circ \notin N$). Thus the rule $a^\bullet \leftarrow a \in \mathrm{TR}(P, Q)_N$ is satisfied in N.

3b. By Lemma 4.5 a rule $a^\circ \leftarrow a \in \mathrm{TR}(P, Q)_N$ ($a \in \mathrm{Hb}(P)$), if $a \notin M'$. If $a \notin N$, the rule is satisfied in N regardless of the satisfiability of a°. If $a \in N$, then $a^\circ \in N$ (since, if $a \notin M'$, then $a^\bullet \notin N$). Thus the rule $a^\circ \leftarrow a \in \mathrm{TR}(P, Q)_N$ is satisfied in N.

3c. By Lemma 4.5 a rule $unsat^\bullet \leftarrow B^\bullet \in \mathrm{TR}(P, Q)_N$, if there exists a rule $A \leftarrow B, {\sim}C \in Q$ such that $M' \not\models \bigvee A$ and $M \models {\sim}C$. Since $M' \models Q_M$, it holds for each rule $A^\bullet \leftarrow B^\bullet \in (Q_M)^\bullet$ that if $(M')^\bullet \not\models \bigvee A^\bullet$, then $(M')^\bullet \not\models B^\bullet$. If $(M')^\bullet \not\models B^\bullet$, then $N \not\models B^\bullet$, since $(M')^\bullet \subseteq N$. Thus each rule of the form $unsat^\bullet \leftarrow B^\bullet \in \mathrm{TR}(P, Q)_N$ is satisfied in N.

3d. By Lemma 4.5 a rule $\mathit{diff} \leftarrow a \in \mathrm{TR}(P,Q)_N$, if there exists $a \in \mathrm{Hb}(P)$ such that $a \notin M'$. Since $\mathit{diff} \in N$, the rule $\mathit{diff} \leftarrow a \in \mathrm{TR}(P,Q)_N$ is satisfied in N regardless of the satisfiability of a.

4, 5, 6. Since $ok \in N$, these are satisfied.

Next we extend the result of the previous lemma and show that the extension $N = \mathrm{EXT}_{P,Q}(M, M')$ is a stable model of the translation $\mathrm{TR}(P,Q)$.

Theorem 4.7. For any two disjunctive logic programs P and Q satisfying $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$, if $\langle M, M'\rangle \in \mathrm{CE}(P,Q)$, then $N = \mathrm{EXT}_{P,Q}(M, M') \in \mathbf{SM}(\mathrm{TR}(P,Q))$ such that $\mathrm{PROJ}_{P,Q}(N) = \langle M, M'\rangle$.

Proof of Theorem 4.7. Suppose that $\langle M, M'\rangle \in \mathrm{CE}(P,Q)$. It follows by Lemma 4.6 that $N \models \mathrm{TR}(P,Q)_N$. To show that $N \in \mathbf{SM}(\mathrm{TR}(P,Q))$ we need to show that $N \in \mathbf{MM}(\mathrm{TR}(P,Q)_N)$. The pairs $\langle M, M'\rangle \in \mathrm{CE}(P,Q)$ divide into two cases corresponding to counter-examples of types (i) T1 and (ii) T2.

(i) Assume that $\langle M, M'\rangle$ is a counter-example of type T1. Thus $M = M'$, and furthermore $M \in \mathbf{SM}(P)$ and $M \not\models Q_M$ (and $M \not\models Q$ by Theorem 2.19). By Definition 4.3, $N = M \cup \{\mathit{unsat}, ok\}$. Assume that $N \notin \mathbf{MM}(\mathrm{TR}(P,Q)_N)$, i.e. there exists $N' \subset N$ such that $N' \models \mathrm{TR}(P,Q)_N$. Thus $N \setminus N' \neq \emptyset$.

- Assume that there exists $a \in \mathrm{Hb}(P)$ such that $a \in N$ and $a \notin N'$. Since $N' \models \mathrm{TR}(P,Q)_N$ and $P_M \subseteq \mathrm{TR}(P,Q)_N$, $N' \models P_M$. Furthermore, $M' = N' \cap \mathrm{Hb}(P) \models P_M$. By Theorem 2.18, this is contradictory to $M \in \mathbf{MM}(P_M)$ since $M' \subset M$. Thus $N \cap \mathrm{Hb}(P) = N' \cap \mathrm{Hb}(P)$.
- Assume that $\mathit{unsat} \notin N'$. Since $M \not\models Q$, there exists a rule $A \leftarrow B, {\sim}C \in Q$ that is not satisfied in M, i.e. $M \not\models \bigvee A$ and $M \models B \cup {\sim}C$. Thus $B \subseteq M \subset N$. Since $\mathit{unsat} \leftarrow B \in \mathrm{TR}(P,Q)_N$ by Lemma 4.5 if $M \not\models \bigvee A$ and $M \models {\sim}C$, to satisfy this rule in N', $B \not\subseteq N'$. This is in contradiction with $N \cap \mathrm{Hb}(P) = N' \cap \mathrm{Hb}(P)$. Thus we must have $\mathit{unsat} \in N'$.
- Assume that $ok \notin N'$. Since $\mathit{unsat} \in N'$, the rule $ok \leftarrow \mathit{unsat} \in \mathrm{TR}(P,Q)_N$ is not satisfied in N'. This is contradictory and thus $ok \in N'$.

Thus $N \in \mathbf{MM}(\mathrm{TR}(P,Q)_N)$, i.e. $N \in \mathbf{SM}(\mathrm{TR}(P,Q))$.

(ii) Assume $\langle M, M'\rangle \in \mathrm{CE}(P,Q)$ is of type T2. Thus $M \in \mathbf{SM}(P)$, $M \models Q_M$, $M' \subset M$ and $M' \models Q_M$. Then by Definition 4.3,

$$N = M \cup \{\mathit{diff}, ok\} \cup \{a^\bullet \mid a \in M'\} \cup \{a^\circ \mid a \in M \setminus M'\}.$$

Let us assume that $N \notin \mathbf{MM}(\mathrm{TR}(P,Q)_N)$, i.e. there exists $N' \subset N$ such that $N' \models \mathrm{TR}(P,Q)_N$. Thus $N \setminus N' \neq \emptyset$.

- Assume that there exists $a \in \mathrm{Hb}(P)$ such that $a \in N$ and $a \notin N'$. Since $N' \models \mathrm{TR}(P,Q)_N$ and by Lemma 4.5 $P_M \subseteq \mathrm{TR}(P,Q)_N$, $N' \models P_M$. Furthermore, $M'' = N' \cap \mathrm{Hb}(P) \models P_M$. By Theorem 2.18, this is in contradiction with $M \in \mathbf{MM}(P_M)$ since $M'' \subset M$. Thus $N \cap \mathrm{Hb}(P) = N' \cap \mathrm{Hb}(P)$.
- Assume that there exists $a \in \mathrm{Hb}(P)$ such that $a^\bullet \in N$ and $a^\bullet \notin N'$. Since $a^\bullet \in N$, by definition of N it holds that $a^\circ \notin N$ ($\Rightarrow a \notin M \setminus M'$) and $a \in N$. Thus, since $unsat \notin N$ and $a \notin M \setminus M'$, by Lemma 4.5 there is a rule $a^\bullet \leftarrow a \in \mathrm{TR}(P,Q)_N$. Since $N \cap \mathrm{Hb}(P) = N' \cap \mathrm{Hb}(P)$, it holds that $a \in N'$. Thus $N' \not\models \mathrm{TR}(P,Q)_N$, which is a contradiction. Therefore $N \cap \mathrm{Hb}(P)^\bullet = N' \cap \mathrm{Hb}(P)^\bullet$.
- Assume that there exists $a \in \mathrm{Hb}(P)$ such that $a^\circ \in N$ and $a^\circ \notin N'$. By the definition of N, it holds that $a \in N$ and $a^\bullet \notin N$ ($\Rightarrow a \notin M'$). Since $a \notin M'$ and $unsat \notin N$, by Lemma 4.5 there is a rule $a^\circ \leftarrow a \in \mathrm{TR}(P,Q)_N$. Since $N \cap \mathrm{Hb}(P) = N' \cap \mathrm{Hb}(P)$, $a \in N'$. Thus $N' \not\models \mathrm{TR}(P,Q)_N$, which is a contradiction. Therefore $N \cap \mathrm{Hb}(P)^\circ = N' \cap \mathrm{Hb}(P)^\circ$.
- Assume that $\mathit{diff} \notin N'$. Since $M' \subset M$, there exists $a \in \mathrm{Hb}(P)$, such that $a \in M$ and $a \notin M'$. Thus $a \in N$. Furthermore, $a \in N'$. Since $unsat \notin N$ and $a \notin M'$, by Lemma 4.5 there is a rule $\mathit{diff} \leftarrow a \in \mathrm{TR}(P,Q)_N$. Since $\mathit{diff} \notin N'$ and $a \in N'$, this rule is not satisfied in N', which is a contradiction. Thus $\mathit{diff} \in N'$.
- Assume that $ok \notin N'$. Since $unsat \notin N$ and $unsat^\bullet \notin N$, by Lemma 4.5 there is a rule $ok \leftarrow \mathit{diff} \in \mathrm{TR}(P,Q)_N$. Since $\mathit{diff} \in N'$ and $ok \notin N'$ this rule is not satisfied in N', which is a contradiction. Thus $ok \in N'$.

Thus $N = N'$, which a contradiction. Therefore it holds that $N \in \mathbf{MM}(\mathrm{TR}(P,Q)_N)$, i.e. $N \in \mathbf{SM}(\mathrm{TR}(P,Q))$.

Finally we need to show that $\mathrm{PROJ}_{P,Q}(N) = \langle M, M' \rangle$. Assume first that $M = M'$. Then $N = M \cup \{unsat, ok\}$ and by Definition 4.4,

$$\mathrm{PROJ}_{P,Q}(N) = \langle N \cap \mathrm{Hb}(P), N \cap \mathrm{Hb}(P) \rangle = \langle M, M \rangle = \langle M, M' \rangle.$$

On the other hand, if $M \neq M'$, then

$$N = M \cup \{\mathit{diff}, ok\} \cup \{a^\bullet \mid a \in M'\} \cup \{a^\circ \mid a \in M \setminus M'\}.$$

By Definition 4.4,

$$\mathrm{PROJ}_{P,Q}(N) = \langle N \cap \mathrm{Hb}(P), \{a \in \mathrm{Hb}(P) \mid a^\bullet \in N\} \rangle = \langle M, M' \rangle. \qquad \square$$

Before discussing the use of the function $\mathrm{PROJ}_{P,Q}$, let us consider the following lemma.

Lemma 4.8. For any disjunctive logic programs P and Q satisfying $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$, if $N \in \mathbf{SM}(\mathrm{TR}(P,Q))$, then $M \in \mathbf{SM}(P)$, where $M = N \cap \mathrm{Hb}(P)$.

Proof of Lemma 4.8. Assume that $N \in \mathbf{SM}(\mathrm{TR}(P,Q))$. First, let us show that $M \models P_M$. Since $N \models \mathrm{TR}(P,Q)_N$ and by Lemma 4.5 $P_M \subseteq \mathrm{TR}(P,Q)_N$, $N \models P_M$. Thus $M = N \cap \mathrm{Hb}(P) \models P_M$.

It also holds that $M \in \mathbf{MM}(P_M)$. Let us assume the opposite, i.e. there exists $M' \subset M$ such that $M' \models P_M$. We define $N' = M' \cup (N \setminus M)$. Thus $N' \subset N$ and $N' \setminus \mathrm{Hb}(P) = N \setminus \mathrm{Hb}(P)$. Let us show that $N' \models \mathrm{TR}(P,Q)_N$.

If we assume that $N' \not\models \mathrm{TR}(P,Q)_N$, there exists a rule in $\mathrm{TR}(P,Q)_N$ that is not satisfied. Using Lemma 4.5, we can examine each part of the translation separately (items corresponding to the items in Lemma 4.5).

1. If a rule $A \leftarrow B \in P_M$ is not satisfied in N', then $N' \not\models P_M$. Thus $M' \not\models P_M$, which is contradictory to $M' \models P_M$.

2. If $unsat \leftarrow B \in \mathrm{TR}(P,Q)_N$ is not satisfied in N', then $B \subseteq N'$ and $unsat \notin N'$. By the definition of N', if $unsat \notin N'$, then $unsat \notin N$. Since $N' \subset N$, also $B \subseteq N$ and $unsat \leftarrow B \in \mathrm{TR}(P,Q)_N$ is not satisfied in N. This is in contradiction with $N \models \mathrm{TR}(P,Q)_N$.

3a. If $a^\bullet \leftarrow a \in \mathrm{TR}(P,Q)_N$ is not satisfied in N', then $a \in N' \subset N$ and $a^\bullet \notin N'$. If $a^\bullet \notin N'$, then $a^\bullet \notin N$. Thus $a^\bullet \leftarrow a \in \mathrm{TR}(P,Q)_N$ is not satisfied in N, which is a contradiction.

3b. If $a^\circ \leftarrow a \in \mathrm{TR}(P,Q)_N$ is not satisfied in N', then $a \in N' \subset N$ and $a^\circ \notin N'$. If $a^\circ \notin N'$, then $a^\circ \notin N$. Thus $a^\circ \leftarrow a \in \mathrm{TR}(P,Q)_N$ is not satisfied in N, which is a contradiction.

3c. If $unsat^\bullet \leftarrow B \in \mathrm{TR}(P,Q)_N$ is not satisfied in N', then $B \subseteq N' \subset N$ and $unsat^\bullet \notin N'$. If $unsat^\bullet \notin N'$, then $unsat^\bullet \notin N$. Thus $unsat \leftarrow B \in \mathrm{TR}(P,Q)_N$ is not satisfied in N, which is a contradiction.

3d. If $diff \leftarrow a \in \mathrm{TR}(P,Q)_N$ is not satisfied in N', then $a \in N' \subset N$ and $diff \notin N'$. If $diff \notin N'$, then $diff \notin N$. Thus $diff \leftarrow a \in \mathrm{TR}(P,Q)_N$ is not satisfied in N, which is a contradiction.

4, 5, 6. Since $N' \setminus \mathrm{Hb}(P) = N \setminus \mathrm{Hb}(P)$, if any of these rules is not satisfied in N', then it is not satisfied in N, which is a contradiction.

Thus the assumption $N' \not\models \mathrm{TR}(P,Q)_N$ leads to contradiction. Therefore $N' \models \mathrm{TR}(P,Q)_N$. Since $N' \subset N$, this is in contradiction with $N \in \mathbf{MM}(\mathrm{TR}(P,Q)_N)$. Thus $M \in \mathbf{MM}(P_M)$, i.e. $M \in \mathbf{SM}(P)$. □

For a stable model N of the translation $\mathrm{TR}(P,Q)$, it holds that $\mathrm{PROJ}_{P,Q}(N)$ is a counter-example for the equivalence of programs P and Q.

Theorem 4.9. For any disjunctive logic programs P and Q satisfying $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$, if $N \in \mathbf{SM}(\mathrm{TR}(P,Q))$, then $\langle M, M' \rangle = \mathrm{PROJ}_{P,Q}(N) \in \mathrm{CE}(P,Q)$ such that $N = \mathrm{EXT}_{P,Q}(M, M')$.

Proof of Theorem 4.9. Assume that $N \in \mathbf{SM}(\mathrm{TR}(P,Q))$. The integrity constraint $\bot \leftarrow \sim ok \in \mathrm{TR}(P,Q)$ forces that ok has to be true in each model of $\mathrm{TR}(P,Q)$, thus it holds that $ok \in N$.

Let $\langle M, M' \rangle = \text{PROJ}_{P,Q}(N)$. Then $M = N \cap \text{Hb}(P)$ and by Lemma 4.8, $M \in \mathbf{SM}(P)$. Thus, to show that $\langle M, M' \rangle \in \text{CE}(P, Q)$, we need to show that $M \notin \mathbf{SM}(Q)$.

Our proof divides into two parts, either (i) $unsat \in N$ (corresponding to a counter-example of type T1) or (ii) $unsat \notin N$ (corresponding to a counter-example of type T2).

(i) Assume that $unsat \in N$. Thus $\text{PROJ}_{P,Q}(N) = \langle M, M' \rangle = \langle M, M \rangle$. Since $N \in \mathbf{SM}(\text{TR}(P, Q))$, $N \in \mathbf{SuppM}(\text{TR}(P, Q))$ by Theorem 2.21. Thus, since $unsat \in N$, there exists a rule $unsat \leftarrow B, {\sim}(A \cup C) \in \text{TR}(P, Q)$ such that $N \models B \cup {\sim}(A \cup C)$. Since $M = N \cap \text{Hb}(P)$, $A \subseteq \text{Hb}(P)$, $B \subseteq \text{Hb}(P)$ and $C \subseteq \text{Hb}(P)$, there exists a rule $A \leftarrow B, {\sim}C \in Q$ such that $M \not\models \bigvee A$, $M \models {\sim}C$ and $M \models B$. It follows that $M \not\models Q$ and furthermore $M \not\models Q_M$ by Theorem 2.19. Thus $\langle M, M \rangle \in \text{CE}(P, Q)$.

(ii) Assume $unsat \notin N$ and thus $\text{PROJ}_{P,Q}(N) = \langle M, M' \rangle$, where $M' = \{a \in \text{Hb}(P) \mid a^\bullet \in N\}$. Assume that $M \not\models Q_M$. Thus there exists a rule $A \leftarrow B \in Q_M$ that is not satisfied in M, i.e. $M \not\models \bigvee A$ and $M \models B$. If $A \leftarrow B \in Q_M$, then there exists a rule $A \leftarrow B, {\sim}C \in Q$, such that $M \models {\sim}C$. Thus by Lemma 4.5 there exists a rule $unsat \leftarrow B \in \text{TR}(P, Q)_N$ that is not satisfied in N, which is a contradiction. Thus $M \models Q_M$.

Since $N \in \mathbf{SM}(\text{TR}(P, Q))$, $N \in \mathbf{SuppM}(\text{TR}(P, Q))$ by Theorem 2.21. Thus, since $ok \in N$, there exists a rule that has the atom ok in its head and its body is satisfied in N. The translation $\text{TR}(P, Q)$ has two rules containing ok in their heads: $ok \leftarrow unsat$ and $ok \leftarrow diff, {\sim}unsat, {\sim}unsat^\bullet$. Since $unsat \notin N$, it must be that $N \models \{diff, {\sim}unsat, {\sim}unsat^\bullet\}$. Thus $unsat^\bullet \notin N$ and $diff \in N$.

Let us show that (a) $M' \models Q_M$ and (b) $M' \subset M$.

(a) Assume that $M' \not\models Q_M$. Thus there exists a rule $A \leftarrow B \in Q_M$ such that $M' \not\models \bigvee A$ and $M' \models B$. If $A \leftarrow B \in Q_M$, then there exists a rule $A \leftarrow B, {\sim}C \in Q$ such that $M \models {\sim}C$. Thus, since $M' \not\models \bigvee A$ and $M \models {\sim}C$, by Lemma 4.5 there exists a rule $unsat^\bullet \leftarrow B^\bullet \in \text{TR}(P, Q)_N$. Since $unsat^\bullet \notin N$, it must hold that $N \not\models B^\bullet$, and furthermore $M' \not\models B$. This is in contradiction with $M' \models B$. Therefore $M' \models Q_M$.

(b) We need to show that if $a \in M'$, then $a \in M$. Let us assume the opposite, i.e. there exists $a \in \text{Hb}(P)$ such that $a \in M'$ and $a \notin M$, i.e. $a^\bullet \in N$ and $a \notin N$. Since $a^\bullet \in N$, the body of the rule $a^\bullet \leftarrow a, {\sim}a^\circ \in \text{TR}(P, Q)$ has to be satisfied in N. Thus $a^\circ \notin N$ and $a \in N$, which is a contradiction. Thus $M' \subseteq M$.

Finally, let us show that $M' \neq M$. As shown earlier, $diff \in N$. Thus, by Theorem 2.21, there exists a rule $diff \leftarrow a, {\sim}a^\bullet \in \text{TR}(P, Q)$ such that $N \models \{a, {\sim}a^\bullet\}$. Hence, there exists $a \in \text{Hb}(P)$ such that $a^\bullet \notin N$ and $a \in N$ and furthermore $a \in M$ and $a \notin M'$.

Since $M' \subseteq M$ and $M' \neq M$, it holds that $M' \subset M$.

Now $M \models Q_M$, $M' \models Q_M$, and $M' \subset M$. Therefore $\langle M, M' \rangle \in \mathrm{CE}(P, Q)$.

Let $N' = \mathrm{EXT}_{P,Q}(M, M')$. We need to show that $N = N'$. Let us first show that $N' \subseteq N$. This divides in two cases.

(i) If $unsat \in N$, then $M = M' = N \cap \mathrm{Hb}(P)$. Since $M = M'$, $N' = \mathrm{EXT}_{P,Q}(M, M) = M \cup \{unsat, ok\} = N \cap \mathrm{Hb}(P) \cup \{unsat, ok\}$. Since it must hold that $ok \in N$ for each stable model of N, $N' \subseteq N$.

(ii) If $unsat \notin N$ then, by Theorem 2.21 and since it must hold that $ok \in N$ for each stable model of N, there must be a rule supporting ok. This implies that either $unsat \in N$ or $\mathit{diff} \in N$. Thus, since $unsat \notin N$, it holds that $\mathit{diff} \in N$. Now,

$$\begin{aligned} N' &= \mathrm{EXT}_{P,Q}(M, M') \\ &= \mathrm{EXT}_{P,Q}(N \cap \mathrm{Hb}(P), \{a \in \mathrm{Hb}(P) \mid a^\bullet \in N\}) \\ &= (N \cap \mathrm{Hb}(P)) \cup \{\mathit{diff}, ok\} \cup (N \cap \mathrm{Hb}(P)^\bullet) \cup (N \cap \mathrm{Hb}(P)^\circ). \end{aligned}$$

Thus it holds that $N' \subseteq N$.

Assume that $N' \subset N$. By Theorem 4.7 it holds that $N' = \mathrm{EXT}_{P,Q}(M, M') \in \mathbf{SM}(\mathrm{TR}(P, Q))$ since $\langle M, M' \rangle \in \mathrm{CE}(P, Q)$. By Theorem 2.18 this is contradictory to the assumption $N \in \mathbf{SM}(\mathrm{TR}(P, Q))$. Thus $N = N'$. □

As a consequence of Theorems 4.7 and 4.9 the functions $\mathrm{EXT}_{P,Q}$ and $\mathrm{PROJ}_{P,Q}$ are bijections when restricted between the sets $\mathbf{SM}(\mathrm{TR}(P, Q))$ and $\mathrm{CE}(P, Q)$. This implies that the counter-examples in $\mathrm{CE}(P, Q)$ and the stable models of the translation $\mathrm{TR}(P, Q)$ are in one-to-one correspondence.

Corollary 4.10. $\mathrm{CE}(P, Q) = \emptyset$ if and only if $\mathbf{SM}(\mathrm{TR}(P, Q)) = \emptyset$.

Furthermore, the relationship in Corollary 4.10 implies the correctness of our method for testing the equivalence of disjunctive logic programs, since $P \equiv Q$ if and only if $\mathrm{CE}(P, Q) = \emptyset$ and $\mathrm{CE}(Q, P) = \emptyset$.

Corollary 4.11. Let P and Q be any disjunctive logic programs such that $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$. $P \equiv Q$ if and only if $\mathbf{SM}(\mathrm{TR}(P, Q)) = \emptyset$ and $\mathbf{SM}(\mathrm{TR}(Q, P)) = \emptyset$.

4.1.2 Computational Complexity Revisited

As discussed in Section 3.2, testing the equivalence of disjunctive programs is $\Pi_2^{\mathbf{P}}$-complete. Let us discuss further how to show that testing the equivalence of disjunctive logic programs is in $\Pi_2^{\mathbf{P}}$ using the translation $\mathrm{TR}(P, Q)$.

In the following we use the notation introduced in Definition 3.13 for languages corresponding to decision problems of our interest. In Section 3.2 we showed that $\overline{\mathrm{IMPR}}$ is in $\Sigma_2^{\mathbf{P}}$ (and thus $\mathrm{IMPR} \in \Pi_2^{\mathbf{P}}$) by giving an abstract algorithm showing how to construct a nondeterministic **NP**-oracle Turing-machine accepting $\overline{\mathrm{IMPR}}$. The translation $\mathrm{TR}(P, Q)$ gives us now means of systematically reducing the problem of equivalence testing of two disjunctive programs into $\overline{\mathrm{SM}}$, which is a problem known to be in $\Pi_2^{\mathbf{P}}$. Thus using the translation as a polynomial-time reduction it is straightforward to show that IMPR is in $\Pi_2^{\mathbf{P}}$.

Theorem 4.12. IMPR is in $\Pi_2^{\mathbf{P}}$.

Proof of Theorem 4.12. Let us take two arbitrary disjunctive logic programs P and Q. It holds that $(P, Q) \in \text{IMPR} \iff \mathbf{SM}(P) \subseteq \mathbf{SM}(Q)$. By Corollary 4.10 this is equivalent to $\text{TR}(P, Q) \notin \text{SM} \iff \text{TR}(P, Q) \in \overline{\text{SM}}$. Since $\overline{\text{SM}} \in \Pi_2^{\mathbf{P}}$ and the reduction from IMPR to $\overline{\text{SM}}$ can be performed in polynomial time using the translation $\text{TR}(P, Q)$, we have $\text{IMPR} \in \Pi_2^{\mathbf{P}}$. □

Similarly, using the translation $\text{TR}(Q, P)$, it can easily be shown that $\text{IMPL} \in \Pi_2^{\mathbf{P}}$, and, furthermore, that $\text{EQV} \in \Pi_2^{\mathbf{P}}$.

4.2 TWO-PHASED TRANSLATION

In the previous section we presented a translation for testing the equivalence of disjunctive logic programs. Finding a counter-example for the equivalence clearly divides in two separate cases (types T1 and T2) and therefore testing can be performed in two phases. The idea is as follows. In the first phase, we use a translation $\text{TR}_1(P, Q)$ to test whether all the stable models of P are also *models* of Q_M. If there exists a stable model of P that is not a model of Q_M, then $\text{TR}_1(P, Q)$ has a stable model and we have found a counter-example of type T1. Otherwise, we will continue to the second phase, where the second translation $\text{TR}_2(P, Q)$ is used to check whether every stable model M of P is a minimal model of Q_M. These two translations can be obtained rather easily by simplifying the previously defined translation $\text{TR}(P, Q)$.

The two-phased approach can be motivated by computational arguments. Counter-examples of type T1 can be found (if there exist any) using a rather straightforward and compact translation, whereas finding a counter-example of type T2 is more complicated. Thus counter-examples of type T2 should be of interest only if counter-examples of type T1 do not exist. In this way, the search space is divided (conditionally) into two parts. The translation for the search of the type T2 counter-examples can be simplified in such an arrangement, since it is known that every stable model M of P is necessarily a model of Q_M.

We use the same notation and new atoms in the definitions of this section as in Definition 4.1 and also assume $\text{Hb}(P) = \text{Hb}(Q)$ similarly as in Section 4.1. We define first the translation $\text{TR}_1(P, Q)$ for the first phase. The Herbrand base of $\text{TR}_1(P, Q)$ contains all the atoms in $\text{Hb}(P)$ and the new atom $unsat$, that is

$$\text{Hb}(\text{TR}_1(P, Q)) = \text{Hb}(P) \cup \{unsat\}.$$

Definition 4.13. Let P and Q be disjunctive logic programs such that $\text{Hb}(P) = \text{Hb}(Q)$. Let $unsat$ be a new atom appearing in neither P nor Q. The translation $\text{TR}_1(P, Q)$ contains the following rules:

1. all the rules of P without modifications,
2. a rule $unsat \leftarrow B, \sim(A \cup C)$ for each rule $A \leftarrow B, \sim C \in Q$, and
3. $\bot \leftarrow \sim unsat$.

The idea behind the translation is as follows. The rules in the first item in Definition 4.13 capture a stable model M of P such that $M = N \cap \text{Hb}(P)$,

where $N \in \mathbf{SM}(\mathrm{TR}_1(P,Q))$. The rules in the second item check whether $M \models Q$ (and thus whether $M \models Q_M$ by Theorem 2.19). Thus, if there exists a rule in Q that is not satisfied in M, $unsat$ is implied. The integrity constraint in the last item ensures that every stable model of the translation contains $unsat$, and therefore a counter-example for the equivalence is found.

We can easily construct a counter-example of type T1 for equivalence from a stable model of the translation $\mathrm{TR}_1(P,Q)$. If $N \in \mathbf{SM}(\mathrm{TR}_1(P,Q))$, then $M \in \mathbf{SM}(P)$ and $M \not\models Q$, where $M = N \cap \mathrm{Hb}(P)$. Thus $M \not\models Q_M$ by Theorem 2.19, and $\langle M, M\rangle$ is a counter-example of type T1.

Example 4.14. Let us consider programs $P = \{a \mid b\}$ and $Q = \{b \leftarrow a\}$. Program P has two stable models, $\mathbf{SM}(P) = \{\{a\},\{b\}\}$, while program Q has one, $\mathbf{SM}(Q) = \{\emptyset\}$. The translation $\mathrm{TR}_1(P,Q)$ is the following:

$$\mathrm{TR}_1(P,Q) = \{a \mid b.\ \ unsat \leftarrow {\sim}b, a.\ \ \bot \leftarrow {\sim}unsat\}.$$

Let us consider a model candidate $N = \{a, unsat\}$. The reduct is

$$\mathrm{TR}_1(P,Q)_N = \{a \mid b.\ \ unsat \leftarrow a\}.$$

Its minimal models are $\mathbf{MM}(\mathrm{TR}_1(P,Q)_N) = \{\{a, unsat\},\{b\}\}$. Thus $N \in \mathbf{SM}(\mathrm{TR}_1(P,Q))$. Since the translation has a stable model, we can conclude that $P \not\equiv Q$. Since $M = N \cap \mathrm{Hb}(P) \in \mathbf{SM}(P)$ and $M \not\models Q_M$, the pair $\langle M, M\rangle$ is a counter-example of type T1. ■

Next, we define the translation $\mathrm{TR}_2(P,Q)$ for the second phase. As mentioned earlier, the use of this translation is needed only if $\mathrm{TR}_1(P,Q)$ has no stable models.

Definition 4.15. Let P and Q be disjunctive logic programs such that $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ and $\mathbf{SM}(\mathrm{TR}_1(P,Q)) = \emptyset$. Let *diff* and $unsat^\bullet$ be new atoms appearing in neither P nor Q. The translation $\mathrm{TR}_2(P,Q)$ contains the following rules:

1. all the rules of P without modifications,
2. rules $a^\bullet \leftarrow a, {\sim}a^\circ$, and $a^\circ \leftarrow a, {\sim}a^\bullet$ for each atom $a \in \mathrm{Hb}(P)$,
3. a rule $unsat^\bullet \leftarrow B^\bullet, {\sim}(A^\bullet \cup C)$ for each rule $A \leftarrow B, {\sim}C \in Q$,
4. a rule $\mathit{diff} \leftarrow a, {\sim}a^\bullet$ for each atom $a \in \mathrm{Hb}(P)$ and
5. rules $\bot \leftarrow {\sim}\mathit{diff}$ and $\bot \leftarrow unsat^\bullet$.

The Herbrand base of $\mathrm{TR}_2(P,Q)$ is the same as $\mathrm{Hb}(\mathrm{TR}(P,Q))$, apart from the atom ok not appearing in $\mathrm{TR}_2(P,Q)$. Thus we have

$$\mathrm{Hb}(\mathrm{TR}_2(P,Q)) = \{\mathit{diff}, unsat^\bullet\} \cup \mathrm{Hb}(P) \cup \mathrm{Hb}(P)^\bullet \cup \mathrm{Hb}(P)^\circ.$$

The idea behind the translation is as follows. The rules in the first item in Definition 4.15 capture a stable model M of P such that $M = N \cap \mathrm{Hb}(P)$, where $N \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$. Since $\mathbf{SM}(\mathrm{TR}_1(P,Q)) = \emptyset$ there is no need to check that $M \models Q_M$ as in Definition 4.1. Thus we can define the translation TR_2 more compactly. As in Definition 4.1, the renamed atoms are used in selecting an interpretation M' for Q_M such that $M' \subseteq M$. The

rules in the second item force that given $a \in M$, either a is true in the submodel candidate ($a^\bullet \in N$) or not ($a^\circ \in N$). The rules in the third item check whether $(M')^\bullet \models (Q_M)^\bullet$, where $(M')^\bullet = N \cap \mathrm{Hb}(P)^\bullet$. Thus, if there exists a rule in Q_M that is not satisfied in M', then $unsat^\bullet$ is implied. The rules in the fourth item check that M' is a proper subset of M, i.e. *diff* is implied if $M' \subset M$. Finally, the integrity constraints in the fifth item are used in ensuring that every stable model of $\mathrm{TR}_2(P, Q)$ contains *diff* but does not contain $unsat^\bullet$.

We can easily construct a counter-example of type T2 for equivalence from a stable model of the translation $\mathrm{TR}_2(P, Q)$. Since $\mathbf{SM}(\mathrm{TR}_1(P, Q)) = \emptyset$, there exists no counter-example of type T1, i.e., if $M \in \mathbf{SM}(P)$, then $M \models Q_M$. Thus, if $N \in \mathbf{SM}(\mathrm{TR}_2(P, Q))$, then $M \in \mathbf{SM}(P)$, $M \models Q_M$ and $M \notin \mathbf{MM}(Q_M)$, where $M = N \cap \mathrm{Hb}(P)$. Moreover, $M' = \{a \mid a^\bullet \in N \cap \mathrm{Hb}(P)^\bullet\} \subset M$ and $M' \models Q_M$. Thus $\langle M, M' \rangle$ is a counter-example of type T2.

Example 4.16. Recall the two logic programs P and Q from Example 4.2, $P = \{a \mid b\}$ and $Q = \{a \leftarrow {\sim}b\}$. Program P has two stable models, $\mathbf{SM}(P) = \{\{a\}, \{b\}\}$, while program Q has one, $\mathbf{SM}(Q) = \{\{a\}\}$. The translation $\mathrm{TR}_1(P, Q)$ is the following,

$$\mathrm{TR}_1(P, Q) = \{a \mid b.\ \ unsat \leftarrow {\sim}a, {\sim}b.\ \ \bot \leftarrow {\sim}unsat\}.$$

The translation $\mathrm{TR}_1(P, Q)$ has no stable models. Thus $\{a\} \models Q$ and $\{b\} \models Q$. The translation $\mathrm{TR}_2(P, Q)$ is presented in Figure 4.2(a). Consider a model candidate $N = \{b, b^\circ, \mathit{diff}\}$. The reduct $\mathrm{TR}_2(P, Q)_N$ is presented in Figure 4.2(b). We have

$$\mathbf{MM}(\mathrm{TR}_2(P, Q)_N) = \{\{a, a^\bullet, a^\circ, \mathit{diff}\}, \{b, b^\circ, \mathit{diff}\}\}.$$

Thus $N \in \mathbf{SM}(\mathrm{TR}_2(P, Q))$. Since the translation has a stable model, we can conclude that $P \not\equiv Q$. The interpretation $M = N \cap \mathrm{Hb}(P) = \{b\}$ is a stable model of P but not a stable model of Q. Thus $M \in \mathbf{SM}(P)$ and $M \notin \mathbf{MM}(Q_M) = \{\emptyset\}$. Moreover, it holds that $M' = \{a \mid a^\bullet \in N \cap \mathrm{Hb}(P)^\bullet\} = \emptyset \subset M$ and $M' \models Q_M$ since $Q_M = \emptyset$. Thus the counter-example $\langle M, M' \rangle$ is of type T2. ■

We denote the use of the two-phased translation on programs P and Q by $[\mathrm{TR}_1(P, Q), \mathrm{TR}_2(P, Q)]$, meaning thereby that the translation $\mathrm{TR}_1(P, Q)$ is used for the first phase and the translation $\mathrm{TR}_2(P, Q)$ is used for the second phase if there is a need for the second phase, i.e. no counter-example of type T1 is found.

4.2.1 Correctness of the Two-Phased Translation

We establish the correctness of the translation $[\mathrm{TR}_1(P, Q), \mathrm{TR}_2(P, Q)]$. Similarly to the case of the one-shot translation $\mathrm{TR}(P, Q)$, there is a tight correspondence between counter-examples for the equivalence and the stable models of the two-phased translation. In this case, we partition the set of counter-examples $\mathrm{CE}(P, Q)$ into $\mathrm{CE}_1(P, Q) \cup \mathrm{CE}_2(P, Q)$ using the type of counter-examples as a criterion, i.e. for any $\langle M, M' \rangle \in \mathrm{CE}(P, Q)$, the pair

$a \mid b.$
$a^\bullet \leftarrow a, {\sim}a^\circ.$
$a^\circ \leftarrow a, {\sim}a^\bullet.$
$b^\bullet \leftarrow b, {\sim}b^\circ.$
$b^\circ \leftarrow b, {\sim}b^\bullet.$
$unsat^\bullet \leftarrow {\sim}a^\bullet, {\sim}b.$
$diff \leftarrow a, {\sim}a^\bullet.$
$diff \leftarrow b, {\sim}b^\bullet.$
$\bot \leftarrow {\sim}diff.$
$\bot \leftarrow unsat^\bullet$

(a)

$a \mid b.$
$a^\bullet \leftarrow a.$
$a^\circ \leftarrow a.$
$b^\circ \leftarrow b.$
$diff \leftarrow a.$
$diff \leftarrow b.$
$\bot \leftarrow unsat^\bullet$

(b)

Figure 4.2: (a) The translation $\mathrm{TR}_2(P, Q)$ from Example 4.16 and (b) the reduct $\mathrm{TR}_2(P, Q)_N$ for $N = \{b, b^\circ, diff\}$.

$\langle M, M' \rangle \in \mathrm{CE}_1(P, Q)$ if $M = M'$, and $\langle M, M' \rangle \in \mathrm{CE}_2(P, Q)$ otherwise. We revise the mappings $\mathrm{EXT}_{P,Q}$ and $\mathrm{PROJ}_{P,Q}$ by dropping the atom "*ok*" that does not appear in $[\mathrm{TR}_1(P, Q), \mathrm{TR}_2(P, Q)]$. The correctness of the two-phased translation is established phase-wise.

Since the translations $\mathrm{TR}_1(P, Q)$ and $\mathrm{TR}_2(P, Q)$ are based on the single translation $\mathrm{TR}(P, Q)$, the correctness proofs for the two-phased translation are very similar to the proofs of correctness of $\mathrm{TR}(P, Q)$. We consider first the translation TR_1. The form of the reduct $\mathrm{TR}_1(P, Q)_N$ for a given interpretation N is as follows.

Lemma 4.17. Given an interpretation N for $\mathrm{TR}_1(P, Q)$, let us define $M = N \cap \mathrm{Hb}(P)$. The reduct $\mathrm{TR}_1(P, Q)_N$ contains the following rules:

1. all the rules of P_M,
2. a rule $unsat \leftarrow B$ if and only if there exists a rule $A \leftarrow B, {\sim}C \in Q$ such that $M \not\models \bigvee A$ and $M \models {\sim}C$, and
3. the rule $\bot$ if and only if $unsat \notin N$.

Lemma 4.17 is proven similarly to Lemma 4.5.

The following lemma states that if there exists $\langle M, M \rangle \in \mathrm{CE}_1(P, Q)$, then $N = \mathrm{EXT}_{P,Q}(M, M)$ is a stable model of the translation $\mathrm{TR}_1(P, Q)$.

Theorem 4.18. For any disjunctive logic programs P and Q satisfying $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$, if $\langle M, M \rangle \in \mathrm{CE}_1(P, Q)$, then $N = \mathrm{EXT}_{P,Q}(M, M) \in \mathbf{SM}(\mathrm{TR}_1(P, Q))$ such that $\mathrm{PROJ}_{P,Q}(N) = \langle M, M \rangle$.

Proof of Theorem 4.18. Assume that $\langle M, M \rangle \in \mathrm{CE}_1(P, Q)$. Thus $M \in \mathbf{SM}(P)$ and $M \not\models Q_M$. Let us first show that $N = \mathrm{EXT}_{P,Q}(M, M) = M \cup \{unsat\} \models \mathrm{TR}_1(P, Q)_N$. Since $M \in \mathbf{SM}(P)$, $M \models P_M$ and furthermore $N \models P_M$ as $M = N \cap \mathrm{Hb}(P)$. Since $unsat \in N$, each rule of the form $unsat \leftarrow B \in \mathrm{TR}_1(P, Q)_N$ is satisfied regardless of the satisfiability of B. Thus $N \models \mathrm{TR}_1(P, Q)_N$.

Let us show that $N \in \mathbf{MM}(\mathrm{TR}_1(P, Q)_N)$. Assume the opposite, i.e. there exists $N' \subset N$ such that $N' \models \mathrm{TR}_1(P, Q)_N$. Thus $N \setminus N' \neq \emptyset$.

- Assume that there exists $a \in \mathrm{Hb}(P)$ such that $a \in N$ and $a \notin N'$. Since $N' \models \mathrm{TR}_1(P,Q)_N$ and $P_M \subseteq \mathrm{TR}_1(P,Q)_N$ by Lemma 4.17, we have $N' \models P_M$. Furthermore, $M' = N' \cap \mathrm{Hb}(P) \models P_M$. By Theorem 2.18, this is contradictory to $M \in \mathbf{MM}(P_M)$ since $M' \subset M$. Thus $N \cap \mathrm{Hb}(P) = N' \cap \mathrm{Hb}(P)$.

- Assume that $unsat \notin N'$. Since $M \not\models Q_M$, $M \not\models Q$ by Theorem 2.19. Thus there exists a rule $A \leftarrow B, {\sim}C \in Q$ that is not satisfied in M, i.e. $M \models {\sim}C$, $M \models B$ and $M \not\models \bigvee A$. Since $B \subseteq M$ and $M = N \cap \mathrm{Hb}(P)$, it holds that $B \subseteq N$. Since by Lemma 4.17 $unsat \leftarrow B \in \mathrm{TR}_1(P,Q)_N$, if $M \not\models \bigvee A$ and $M \models {\sim}C$, to satisfy this rule in N', $B \not\subseteq N'$ must hold. This is in contradiction with $N \cap \mathrm{Hb}(P) = N' \cap \mathrm{Hb}(P)$.

Thus $N \in \mathbf{MM}(\mathrm{TR}_1(P,Q)_N)$, i.e. $N \in \mathbf{SM}(\mathrm{TR}_1(P,Q))$.

Finally, it holds that $\mathrm{PROJ}_{P,Q}(N) = \langle M, M \rangle$, since $unsat \in N$ and $M = N \cap \mathrm{Hb}(P)$. □

On the other hand, if $N \in \mathbf{SM}(\mathrm{TR}_1(P,Q))$, then $\mathrm{PROJ}_{P,Q}(N)$ is a counter-example of type T1.

Theorem 4.19. For any disjunctive logic programs P and Q satisfying $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$, if $N \in \mathbf{SM}(\mathrm{TR}_1(P,Q))$, then $\langle M, M \rangle = \mathrm{PROJ}_{P,Q}(N) \in \mathrm{CE}_1(P,Q)$ such that $N = \mathrm{EXT}_{P,Q}(M,M)$.

Proof of Theorem 4.19. Assume that $N \in \mathbf{SM}(\mathrm{TR}_1(P,Q))$. By Definition 4.4 $\mathrm{PROJ}_{P,Q}(N) = \langle M, M \rangle$, where $M = N \cap \mathrm{Hb}(P)$. First, let us show that $M \models P_M$. Lemma 4.17 gives us the form of the reduct $\mathrm{TR}_1(P,Q)_N$. Since $N \models \mathrm{TR}_1(P,Q)_N$ and $P_M \subseteq \mathrm{TR}_1(P,Q)_N$ by Lemma 4.17, $N \models P_M$. Thus $M = N \cap \mathrm{Hb}(P) \models P_M$.

It also holds that $M \in \mathbf{MM}(P_M)$. Let us assume the opposite, i.e. there exists $M' \subset M$ such that $M' \models P_M$. Let us define $N' = M' \cup (N \setminus M)$. Thus $N' \subset N$ and $N' \setminus \mathrm{Hb}(P) = N \setminus \mathrm{Hb}(P)$. Now, it holds that $N' \models \mathrm{TR}_1(P,Q)_N$.

If we assume $N' \not\models \mathrm{TR}_1(P,Q)_N$, there exists a rule in $\mathrm{TR}_1(P,Q)_N$ that is not satisfied in N'.

- If a rule $A \leftarrow B \in P_M$ is not satisfied in N', then $N' \not\models P_M$. Thus $M' = N' \cap \mathrm{Hb}(P) \not\models P_M$, which is a contradiction.

- If $unsat \leftarrow B \in \mathrm{TR}_1(P,Q)_N$ is not satisfied in N', then $B \subseteq N'$ and $unsat \notin N'$. By the definition of N', if $unsat \notin N'$, then $unsat \notin N$. If $unsat \notin N$, then by Lemma 4.17 $\bot \in \mathrm{TR}_1(P,Q)_N$ and $N \not\models \mathrm{TR}_1(P,Q)_N$, which is a contradiction.

Thus the assumption $N' \not\models \mathrm{TR}_1(P,Q)_N$ leads to a contradiction and therefore $N' \models \mathrm{TR}_1(P,Q)_N$. Since $N' \subset N$, this is in contradiction with $N \in \mathbf{MM}(\mathrm{TR}_1(P,Q)_N)$ by Theorem 2.18. Thus $M \in \mathbf{MM}(P_M)$, i.e. $M \in \mathbf{SM}(P)$.

We need to show that $M \not\models Q_M$. The integrity constraint $\bot \leftarrow {\sim}unsat$ forces $unsat \in N$ for each $N \in \mathbf{SM}(\mathrm{TR}_1(P,Q))$. Furthermore, since $N \in \mathbf{SM}(\mathrm{TR}_1(P,Q))$, by Theorem 2.21 we have $N \in \mathbf{SuppM}(\mathrm{TR}_1(P,Q))$.

Thus, there exists a rule $unsat \leftarrow B, {\sim}(A \cup C) \in \mathrm{TR}_1(P,Q)$ such that $N \models B$ and $N \models {\sim}(A \cup C)$. Thus, since $M = N \cap \mathrm{Hb}(P)$, there exists a rule $A \leftarrow B, {\sim}C \in Q$ such that $M \not\models \bigvee A$, $M \models {\sim}C$ and $M \models B$. This implies that $M \not\models Q$. Furthermore, $M \not\models Q_M$ by Theorem 2.19. Since $M \in \mathbf{SM}(P)$ and $M \not\models Q_M$, it holds that $\langle M, M\rangle \in \mathrm{CE}_1(P,Q)$.

Finally, let $N' = \mathrm{EXT}_{P,Q}(M,M)$. We need to show that $N = N'$. By Definition 4.3, $N' = \mathrm{EXT}_{P,Q}(M,M) = M \cup \{unsat\} = (N \cap \mathrm{Hb}(P)) \cup \{unsat\}$. Thus $N' \subseteq N$. Since $\mathrm{Hb}(\mathrm{TR}_1(P,Q)) = \mathrm{Hb}(P) \cup \{unsat\}$, and $unsat \in N$, it holds that $N' = N$. □

As a consequence of Theorems 4.18 and 4.19, the counter-examples of type T1 and the stable models of the translation $\mathrm{TR}_1(P,Q)$ are in one-to-one correspondence.

Corollary 4.20. $\mathrm{CE}_1(P,Q) = \emptyset$ if and only if $\mathbf{SM}(\mathrm{TR}_1(P,Q)) = \emptyset$.

Next, we consider the second phase of the two-phased translation. The form of the reduct $\mathrm{TR}_2(P,Q)_N$ for a given interpretation N is as follows.

Lemma 4.21. Given an interpretation N for $\mathrm{TR}_2(P,Q)$, let us define $M_1 = N \cap \mathrm{Hb}(P)$, $M_2 = \{a \in \mathrm{Hb}(Q) \mid a^\bullet \in N\}$ and $R = \{a \in \mathrm{Hb}(Q) \mid a^\circ \in N\}$. Thus ${M_2}^\bullet = N \cap \mathrm{Hb}(Q)^\bullet$ and $R^\circ = N \cap \mathrm{Hb}(Q)^\circ$. The reduct $\mathrm{TR}_2(P,Q)_N$ contains the following rules:

1. all the rules of P_{M_1},
2. a rule $a^\bullet \leftarrow a$ if and only if there exists $a \in \mathrm{Hb}(P)$ such that $a \notin R$, and
   a rule $a^\circ \leftarrow a$ if and only if there exists $a \in \mathrm{Hb}(P)$ such that $a \notin M_2$,
3. a rule $unsat^\bullet \leftarrow B^\bullet$ if and only if there exists a rule $A \leftarrow B, {\sim}C \in Q$ such that $M_2 \not\models \bigvee A$ and $M_1 \models {\sim}C$,
4. a rule $diff \leftarrow a$ if and only if there exists $a \in \mathrm{Hb}(P)$ such that $a \notin M_2$, and
5. the rule $\bot$ if and only if $diff \notin N$ and the rule $\bot \leftarrow unsat^\bullet$.

Lemma 4.21 is proven similarly to Lemma 4.5.

If there is no counter-example of type T1, then, if there exists a counter-example $\langle M, M'\rangle \in \mathrm{CE}_2(P,Q)$, we have that $N = \mathrm{EXT}_{P,Q}(M,M')$ is a stable model of $\mathrm{TR}_2(P,Q)$.

Theorem 4.22. For any disjunctive logic programs P and Q satisfying $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ and $\mathbf{SM}(\mathrm{TR}_1(P,Q)) = \emptyset$, if $\langle M, M'\rangle \in \mathrm{CE}_2(P,Q)$, then $N = \mathrm{EXT}_{P,Q}(M,M') \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$ such that $\mathrm{PROJ}_{P,Q}(N) = \langle M, M'\rangle$.

Proof of Theorem 4.22. Since $\mathbf{SM}(\mathrm{TR}_1(P,Q)) = \emptyset$, we have $\mathrm{CE}_1(P,Q) = \emptyset$ by Corollary 4.20. Furthermore, for each $M \in \mathbf{SM}(P)$ it holds that $M \models Q_M$. Assume $\langle M, M'\rangle \in \mathrm{CE}_2(P,Q)$, i.e. $M' \subset M$ and $M' \models Q_M$. We have

$$N = \mathrm{EXT}_{P,Q}(M,M') = M \cup \{a^\bullet \mid a \in M'\} \cup \{a^\circ \mid a \in M \setminus M'\} \cup \{diff\}.$$

Let us show that each rule in the reduct $\mathrm{TR}_2(P,Q)_N$ is satisfied in N (items corresponding to the items in Lemma 4.21 such that $M_1 = M$, $M_2 = M'$ and $R = M \setminus M'$).

1. $M \models P_M$ and thus $N \models P_M$.

2. By Lemma 4.21 a rule $a^\bullet \leftarrow a \in \mathrm{TR}_2(P,Q)_N$ ($a \in \mathrm{Hb}(P)$), if $a \notin M \setminus M'$. If $a \notin N$, the rule is satisfied in N regardless of the satisfiability of $a^\bullet$. If $a \in N$, then $a^\bullet \in N$ (since $a \notin M \setminus M'$, it holds that $a^\circ \notin N$). Thus the rule $a^\bullet \leftarrow a \in \mathrm{TR}_2(P,Q)_N$ is satisfied in N.

   By Lemma 4.21 a rule $a^\circ \leftarrow a \in \mathrm{TR}_2(P,Q)_N$ ($a \in \mathrm{Hb}(P)$), if $a \notin M'$. If $a \notin N$, the rule is satisfied in N regardless of the satisfiability of a°. If $a \in N$, then $a^\circ \in N$ (since $a \notin M'$, it holds that $a^\bullet \notin N$). Thus the rule $a^\circ \leftarrow a \in \mathrm{TR}_2(P,Q)_N$ is satisfied in N.

3. By Lemma 4.21 a rule $unsat^\bullet \leftarrow B^\bullet \in \mathrm{TR}_2(P,Q)_N$, if there exists a rule $A \leftarrow B, {\sim}C \in Q$ such that $M' \not\models \bigvee A$ and $M \models {\sim}C$. Assume that a rule $unsat^\bullet \leftarrow B^\bullet \in \mathrm{TR}_2(P,Q)_N$ is not satisfied in N. Then $N \models B^\bullet$ which implies that $(M')^\bullet \models B^\bullet$ and furthermore $M' \models B$. Thus a rule $A \leftarrow B \in Q_M$ is not satisfied in M', which is in contradiction with $M' \models Q_M$. Thus each rule of the form $unsat^\bullet \leftarrow B^\bullet \in \mathrm{TR}_2(P,Q)_N$ is satisfied in N.

4. Since $\mathit{diff} \in N$, each rule of the form $\mathit{diff} \leftarrow a \in \mathrm{TR}_2(P,Q)_N$ is satisfied regardless of the satisfiability of $a \in \mathrm{Hb}(P)$.

5. Since $\mathit{diff} \in N$ and $unsat^\bullet \notin N$, these rules are satisfied.

Thus $N \models \mathrm{TR}_2(P,Q)_N$.

Next we show that $N \in \mathbf{MM}(\mathrm{TR}_2(P,Q)_N)$. Assume the opposite, i.e. there exists $N' \subset N$ such that $N' \models \mathrm{TR}_2(P,Q)_N$. Thus $N \setminus N' \neq \emptyset$.

- Assume that there exists $a \in \mathrm{Hb}(P)$ such that $a \in N$ and $a \notin N'$. Since $N' \models \mathrm{TR}_2(P,Q)_N$ and by Lemma 4.21 $P_M \subseteq \mathrm{TR}_2(P,Q)_N$, $N' \models P_M$. Furthermore, $M'' = N' \cap \mathrm{Hb}(P) \models P_M$. since $M'' \subset M$, this is in contradiction with $M \in \mathbf{MM}(P_M)$ by Theorem 2.18. Thus $N \cap \mathrm{Hb}(P) = N' \cap \mathrm{Hb}(P)$.

- Assume that there exists $a \in \mathrm{Hb}(P)$ such that $a^\bullet \in N$ and $a^\bullet \notin N'$. Since $a^\bullet \in N$, $a^\circ \notin N$ ($\Rightarrow a \notin M \setminus M'$) and $a \in N$. Since $a \notin M \setminus M'$, by Lemma 4.21 there is a rule $a^\bullet \leftarrow a \in \mathrm{TR}_2(P,Q)_N$. Since $N \cap \mathrm{Hb}(P) = N' \cap \mathrm{Hb}(P)$, $a \in N'$. Thus $N' \not\models \mathrm{TR}_2(P,Q)_N$, which is a contradiction. Therefore $N \cap \mathrm{Hb}(P)^\bullet = N' \cap \mathrm{Hb}(P)^\bullet$.

- Assume that there exists $a \in \mathrm{Hb}(P)$ such that $a^\circ \in N$ and $a^\circ \notin N'$. By the definition of N, $a \in N$ and $a^\bullet \notin N$ ($\Rightarrow a \notin M'$). Since $a \notin M'$, by Lemma 4.21 there is a rule $a^\circ \leftarrow a \in \mathrm{TR}_2(P,Q)_N$. Since $N \cap \mathrm{Hb}(P) = N' \cap \mathrm{Hb}(P)$, $a \in N'$. Thus $N' \not\models \mathrm{TR}_2(P,Q)_N$, which is a contradiction. Therefore $N \cap \mathrm{Hb}(P)^\circ = N' \cap \mathrm{Hb}(P)^\circ$.

- Assume that $\mathit{diff} \notin N'$. Since $M' \subset M$, there exists $a \in \mathrm{Hb}(P)$ such that $a \in M$ and $a \notin M'$. Thus $a \in N$ ($\Rightarrow a \in N'$) and $a^\bullet \notin N$ ($\Rightarrow a \notin M'$). Since $a \notin M'$, by Lemma 4.21 there is a rule $\mathit{diff} \leftarrow a \in \mathrm{TR}_2(P,Q)_N$. Since $\mathit{diff} \notin N'$, this rule is not satisfied in N', which is a contradiction. Thus $\mathit{diff} \in N'$.

Thus $N = N'$ and so $N \in \mathbf{MM}(\mathrm{TR}_2(P,Q)_N)$, i.e. $N \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$.

Finally, $\mathrm{PROJ}_{P,Q}(N) = \langle M, M' \rangle$, since $M = N \cap \mathrm{Hb}(P)$ and $M' = \{a \in \mathrm{Hb}(P) \mid a^\bullet \in N\}$. □

If there exists no counter-examples of type T1, then a stable model N of the translation $\mathrm{TR}_2(P,Q)$ corresponds to a counter-example of type T2, i.e. it holds that $\mathrm{PROJ}_{P,Q}(N) \in \mathrm{CE}_2(P,Q)$.

Theorem 4.23. For any disjunctive logic programs P and Q satisfying $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ and $\mathbf{SM}(\mathrm{TR}_1(P,Q)) = \emptyset$, if $N \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$, then $\langle M, M' \rangle = \mathrm{PROJ}_{P,Q}(N) \in \mathrm{CE}_2(P,Q)$ such that $N = \mathrm{EXT}_{P,Q}(M, M')$.

Proof of Theorem 4.23. Let $N \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$. First, we will show that $M \models P_M$. Since $N \models \mathrm{TR}_2(P,Q)_N$ and by Lemma 4.21 $P_M \subseteq \mathrm{TR}_2(P,Q)_N$, $N \models P_M$. Thus $M = N \cap \mathrm{Hb}(P) \models P_M$.

It also holds that M is a minimal model of P_M. Let us assume the opposite, i.e. there exists $M' \subset M$ such that $M' \models P_M$. Let $N' = M' \cup (N \setminus M)$. Thus $N' \subset N$ and $N' \setminus \mathrm{Hb}(P) = N \setminus \mathrm{Hb}(P)$, and we have $N' \models \mathrm{TR}_2(P,Q)_N$.

If we assume the opposite, i.e. $N' \not\models \mathrm{TR}_2(P,Q)_N$, there exists a rule in $\mathrm{TR}_2(P,Q)_N$ that is not satisfied in N'. Following items correspond to the items in Lemma 4.21 respectively.

1. If a rule $A \leftarrow B \in P_M$ is not satisfied in N', then $N' \not\models P_M$. Thus $M' \not\models P_M$, which is a contradiction.

2. If $a^\bullet \leftarrow a \in \mathrm{TR}_2(P,Q)_N$ is not satisfied in N', then $a \in N' \subset N$ and $a^\bullet \notin N'$ ($\Rightarrow a^\bullet \notin N$). Thus $a^\bullet \leftarrow a \in \mathrm{TR}_2(P,Q)_N$ is not satisfied in N, which is a contradiction.

   If $a^\circ \leftarrow a \in \mathrm{TR}_2(P,Q)_N$ is not satisfied in N', then $a \in N' \subset N$ and $a^\circ \notin N'$ ($\Rightarrow a^\circ \notin N$). Thus $a^\circ \leftarrow a \in \mathrm{TR}_2(P,Q)_N$ is not satisfied in N, which is a contradiction.

3. If $unsat^\bullet \leftarrow B^\bullet \in \mathrm{TR}_2(P,Q)_N$ is not satisfied in N', then $B^\bullet \subseteq N' \subset N$ and $unsat^\bullet \notin N'$ ($\Rightarrow unsat^\bullet \notin N$). Thus $unsat^\bullet \leftarrow B^\bullet \in \mathrm{TR}_2(P,Q)_N$ is not satisfied in N, which is a contradiction.

4. If $\mathit{diff} \leftarrow a \in \mathrm{TR}_2(P,Q)_N$ is not satisfied in N', then $a \in N' \subset N$ and $\mathit{diff} \notin N'$ ($\Rightarrow \mathit{diff} \notin N$). Thus $\mathit{diff} \leftarrow a \in \mathrm{TR}_2(P,Q)_N$ is not satisfied in N, which is a contradiction.

5. Since $N' \setminus \mathrm{Hb}(P) = N \setminus \mathrm{Hb}(P)$, if these rules are not satisfied in N', then they are not satisfied in N, which is a contradiction.

Thus the assumption $N' \not\models \mathrm{TR}_2(P,Q)_N$ leads to a contradiction. Therefore $N' \models \mathrm{TR}_2(P,Q)_N$. Since $N' \subset N$, this is in contradiction with $N \in \mathbf{MM}(\mathrm{TR}_2(P,Q)_N)$. Thus $M \in \mathbf{MM}(P_M)$, i.e. $M \in \mathbf{SM}(P)$.

We need to show that $M \notin \mathbf{MM}(Q_M)$. Since $\mathbf{SM}(\mathrm{TR}_1(P,Q)) = \emptyset$ by Theorem 4.19, it holds that $M \models Q_M$. The integrity constraint $\bot \leftarrow unsat^\bullet \in \mathrm{TR}_2(P,Q)$ forces that $unsat^\bullet$ cannot be true in any model of $\mathrm{TR}_2(P,Q)$ and the integrity constraint $\bot \leftarrow {\sim}\mathit{diff} \in \mathrm{TR}_2(P,Q)$ forces that *diff* is true in each model of $\mathrm{TR}_2(P,Q)$.

We show that (i) $M' \models Q_M$, (ii) $M' \subseteq M$ and (iii) $M \neq M'$.

(i) Let us assume the opposite, $M' \not\models Q_M$, i.e. there exists a rule $A \leftarrow B \in Q_M$ that is not satisfied in M'. Thus $M' \not\models \bigvee A$ and $M' \models B$. The rule $A \leftarrow B \in Q_M$, if $A \leftarrow B, {\sim}C \in Q$ and $M \models {\sim}C$. Since $M' \not\models \bigvee A$ and $M \models {\sim}C$ there exists a rule $unsat^\bullet \leftarrow B^\bullet \in \mathrm{TR}_2(P,Q)_N$ by Lemma 4.21. The rule $unsat^\bullet \leftarrow B^\bullet$ is not satisfied in N, since $(M')^\bullet \models B^\bullet$ ($\Rightarrow N \models B^\bullet$) and $unsat^\bullet \notin N$. This is in contradiction with $N \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$. Therefore $M' \models Q_M$.

(ii) We need to show that if $a \in M'$, then $a \in M$. Let us assume the opposite, i.e. there exists $a \in \mathrm{Hb}(P)$ such that $a \in M'$ ($\Rightarrow a^\bullet \in N$) and $a \notin M$ ($\Rightarrow a \notin N$).

Since $a^\bullet \in N$ and the rule $a^\bullet \leftarrow a, {\sim}a^\circ$ is the only rule in $\mathrm{TR}_2(P,Q)$ containing the atom $a^\bullet$ in its head, the body of the rule $a^\bullet \leftarrow a, {\sim}a^\circ \in \mathrm{TR}_2(P,Q)$ must be satisfied in N by Theorem 2.21. Thus $a^\circ \notin N$ and $a \in N$, which is in contradiction with the assumption $a \notin N$. Therefore $M' \subseteq M$.

(iii) Since $N \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$, $N \in \mathbf{SuppM}(\mathrm{TR}_2(P,Q))$ by Theorem 2.21. Thus, since $diff \in N$, there exists a rule $diff \leftarrow a, {\sim}a^\bullet \in \mathrm{TR}_2(P,Q)$ such that $N \models \{a, {\sim}a^\bullet\}$. Thus there exists $a \in \mathrm{Hb}(P)$ such that $a^\bullet \notin N$ and $a \in N$. Furthermore, $a \in M$ and $a \notin M'$. Thus, $M' \neq M$.

Now, $M \models Q_M$, $M' \models Q_M$, and $M' \subset M$. Therefore $M \notin \mathbf{MM}(Q_M)$ and $\langle M, M' \rangle \in \mathrm{CE}_2(P,Q)$.

Finally, we need to show that $N = N'$, where $N' = \mathrm{EXT}_{P,Q}(M, M')$. Since $N \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$ it holds that $unsat^\bullet \notin N$ and $diff \in N$. We have

$$\begin{aligned} N' &= \mathrm{EXT}_{P,Q}(M, M') \\ &= \mathrm{EXT}_{P,Q}(N \cap \mathrm{Hb}(P), \{a \in \mathrm{Hb}(P) \mid a^\bullet \in N\}) \\ &= (N \cap \mathrm{Hb}(P)) \cup \{diff\} \cup (N \cap \mathrm{Hb}(P)^\bullet) \cup (N \cap \mathrm{Hb}(P)^\circ). \end{aligned}$$

Thus it holds that $N' \subseteq N$. Let us assume that $N' \subset N$. By Theorem 4.22 it holds that $N' = \mathrm{EXT}_{P,Q}(M, M') \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$, since $\langle M, M' \rangle \in \mathrm{CE}_2(P,Q)$. By Theorem 2.18 this is contradictory to the assumption $N \in \mathbf{SM}(\mathrm{TR}_2(P,Q))$ and thus $N = N'$. □

As a consequence of Theorems 4.22 and 4.23 counter-examples of type T2 and the stable models of the translation $\mathrm{TR}_2(P,Q)$ are in one-to-one correspondence.

Corollary 4.24. Assume $\mathbf{SM}(\mathrm{TR}_1(P,Q)) = \emptyset$. Then $\mathrm{CE}_2(P,Q) = \emptyset$ if and only if $\mathbf{SM}(\mathrm{TR}_2(P,Q)) = \emptyset$.

Now, using Corollaries 4.20 and 4.24 we can test the equivalence of disjunctive logic programs P and Q using the two translations $\mathrm{TR}_1(P,Q)$ and $\mathrm{TR}_2(P,Q)$. For convenience, we define the stable models of the two-phased translation $[\mathrm{TR}_1(P,Q), \mathrm{TR}_2(P,Q)]$ as a slight notational generalization.

Definition 4.25. Let P and Q be disjunctive logic programs such that $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$. We define the stable models of the two-phased translation $[\mathrm{TR}_1(P,Q), \mathrm{TR}_2(P,Q)]$ as follows.

$$\mathbf{SM}([\mathrm{TR}_1(P,Q), \mathrm{TR}_2(P,Q)]) = \begin{cases} \mathbf{SM}(\mathrm{TR}_1(P,Q)), & \text{if } \mathbf{SM}(\mathrm{TR}_1(P,Q)) \neq \emptyset, \\ \mathbf{SM}(\mathrm{TR}_2(P,Q)), & \text{otherwise.} \end{cases}$$

Corollary 4.26. $P \equiv Q$ if and only if $\mathbf{SM}([\mathrm{TR}_1(P,Q), \mathrm{TR}_2(P,Q)]) = \emptyset$ and $\mathbf{SM}([\mathrm{TR}_1(Q,P), \mathrm{TR}_2(Q,P)]) = \emptyset$, where P and Q are any disjunctive logic programs such that $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$.

5 EXPERIMENTS

In this chapter we present experiments which compare the performance of the implementation of the two translations presented in Chapter 4 with a fictitious naive approach of cross-checking the stable models. First, in Section 5.1, we present the current implementation of translations TR and $[\mathrm{TR}_1, \mathrm{TR}_2]$. The naive approach is presented in Section 5.2. The test arrangements are discussed in Section 5.3 and the results of the experiments are reported in Sections 5.4 and 5.5.

5.1 IMPLEMENTATION

Translation functions $\mathrm{TR}(P, Q)$ and $[\mathrm{TR}_1(P, Q), \mathrm{TR}_2(P, Q)]$, as presented in Chapter 4, have been implemented in C under Linux. The translator called DLPEQ [30] takes two disjunctive logic programs P and Q as its input and produces by default the translation $\mathrm{TR}(P, Q)$ as its output. The input files are assumed to be in the internal format of SMODELS/GNT produced by the front-end LPARSE. The implementation DLPEQ supports programs containing disjunctive rules and, in addition, *compute statements*. Compute statements can be used to define additional constraints for stable models of the program. Rules with variables can also be used, although the front-end LPARSE performs an instantiation for the rules.

The GNT system [19] is an experimental implementation of the stable model semantics for disjunctive logic programs. The implementation is based on an architecture consisting of two interacting SMODELS solvers [37] for non-disjunctive programs. One of them is responsible for generating the possible model candidates while the other checks for minimality, as required from disjunctive stable models.

Let us consider a detailed example of how to use the programs LPARSE, GNT and DLPEQ to compute stable models and test the equivalence of disjunctive logic programs.

Example 5.1. The rules are presented in the following syntax

```
a1 | a2 | a3 :- b1, b2, b3, not c1, not c2, not c3.
```

and (possible) compute statements as

```
compute {a1, a2, not b1, not b2}.
```

Assume that file `input.lp` contains a disjunctive program `a | b.` For disjunctive logic programs the front-end LPARSE is invoked using command line option `--dlp`. LPARSE produces the logic program in the internal format of GNT.

```
$ lparse --dlp input.lp
3 2 1 2 0 0
0
1 b
```

```
2 a
0
B+
0
B-
0
1
```

If the file `input.sm` contains a disjunctive logic program in the internal format of GNT, all its stable models can be computed as follows.

```
$ more input.sm | gnt 0
Answer set: 1
Stable Model: a
Answer set: 2
Stable Model: b
False
Number of isminimal calls: 2
Duration: 0.000
```

If `0` is replaced by a positive integer n, then up to n stable models are computed.

Now, assume that the files `input1.sm` and `input2.sm` contain logic programs $P = \{a \mid b\}$ and $Q = \{a \leftarrow \sim b\}$ in the internal format of GNT, respectively. We produce the translation $\mathrm{TR}(P, Q)$ using DLPEQ and compute *one* stable model using GNT as follows.

```
$ dlpeq input1.sm input2.sm | gnt 1
Answer set: 1
Stable Model: b b''
True
Number of isminimal calls: 1
Duration: 0.000
```

Thus the translation $\mathrm{TR}(P, Q)$ has a stable model and we have $P \not\equiv Q$. ■

Translations $\mathrm{TR}_1(P, Q)$ and $\mathrm{TR}_2(P, Q)$ can be produced using the command line option `-p [1|2]`. The translation can be printed in a textual form using option `-v`. Notice that the new atoms $unsat$, $unsat^\bullet$, $diff$ and ok are presented in the form `_n` in the textual presentation, where `n` is an integer. The renamed atoms $a^\bullet$ and a° are presented as `a'` and `a''`, respectively.

DLPEQ is designed to be used with the solver GNT. The current implementation also enables the use of another state of the art solver, namely DLV [6].

Example 5.2. Programs `input1.sm` and `input2.sm` are translated using DLPEQ with the option `--dlv` and DLV is used to compute one stable model.

```
$ dlpeq --dlv input1.sm input2.sm | dlv -silent -n=1 --
{b, b__, a_9, a_10}
```

Notice that the presentation of the new atoms is different from the normal verbose mode of DLPEQ. The new atoms $unsat$, $unsat^{\bullet}$, $diff$ and ok are presented in the form `a_n` (`n` is an integer) in the DLV presentation, and the renamed atoms $a^{\bullet}$ and a° as `a_` and `a__`, respectively. ■

To summarize the different features, the usage of DLPEQ is presented in Figure 5.1.

```
usage:   dlpeq <options> <file1> <file2>

options:
   -h or --help -- print help message
   --version -- print version information
   -p [1|2] or --phase [1|2]  -- two-phased translation,
                                 phase 1 or 2
   -v -- verbose mode (human readable)
   --dlv -- verbose mode, dlv syntax
```

Figure 5.1: Usage of DLPEQ.

It is important to note that DLPEQ checks that the *visible* Herbrand bases of the disjunctive logic programs being compared are exactly the same. A visible atom has a name in the symbol table of the program. However, when integrity constraints are used, the front-end LPARSE may produce some *invisible* atoms that are not taken into account in this comparison. To support programs produced by LPARSE, such atoms keep their roles in the respective programs.

5.2 THE NAIVE APPROACH

To assess the feasibility of DLPEQ in practice we ran tests to compare running times of the DLPEQ (one-shot translation) and the DLPEQ-2 (two-phased translation) approach with those of a naive approach. The NAIVE approach involves computing all stable models of one program and checking that they also are stable models of the other program, and vice versa. This idea is formalized in the procedure **TestNaive** presented in Figure 5.2. We can present the NAIVE approach to equivalence testing as follows.

Definition 5.3. Given two disjunctive logic programs P and Q, to establish the equivalence of the programs one needs to run **TestNaive**(P, Q) and **TestNaive**(Q, P). If both return "Success", P and Q are equivalent, otherwise they are not.

The NAIVE approach is implemented as a shell script. Testing that the stable models of P are also stable models of Q is realized in practice by adding a compute statement of the form 'compute $\{a_1, \ldots, a_n, \sim b_1, \ldots, \sim b_m\}$' to Q, where $a_1, \ldots, a_n \in M$ and $b_1, \ldots, b_m \in \mathrm{Hb}(P) \setminus M$. In this way, the model candidate M for Q is already fixed when stable models of Q are searched and thus only the minimality condition essential to stable models needs to be checked.

```
TestNaive(P, Q)
1  for each M ∈ SM(P)
2  do
3     if M ∉ SM(Q)
4        then return Failure
5     fi
6  done
7  return Success
```

Figure 5.2: The algorithm for the NAIVE approach in equivalence testing.

The NAIVE approach is naive in the sense that there is no internal mechanism to guide the search for counter-examples for the equivalence. We believe the situation to be different when the translation-based approaches are used. With the NAIVE approach one has to check all the stable models of both programs in the worst case, which can become very time-consuming. Thus, we assume that the translation-based approaches are to perform better than the NAIVE approach if the programs to be compared have many stable models. However, if the programs to be tested have very few or no stable models it is likely that the NAIVE approach is faster than the translation-based approaches.

5.3 TEST ARRANGEMENTS

The NAIVE approach for testing the equivalence of disjunctive programs P and Q is formalized in Definition 5.3 and the approach always involves two directions. Similarly, in the DLPEQ and DLPEQ-2 approaches, to verify the equivalence one checks the stable models of the translations $\mathrm{TR}(P, Q)$ and $[\mathrm{TR}_1(P, Q), \mathrm{TR}_2(P, Q)]$ in both directions.

There is, however, still room for optimization in all the approaches. If one finds a counter-example in one direction, then $P \not\equiv Q$ holds and there is no need to test the other direction, except if one wishes to perform a thorough analysis. We decided not to use this optimization, since running times turned out to scale differently depending on the chosen direction. Thus the running times reported always involve sum of the running times of both directions.

With all the three approaches, the GNT system is responsible for computation of the stable models. In the DLPEQ approach, the total running time (in one direction) is the running time needed for GNT for trying to compute one stable model of the translation TR produced by DLPEQ. In the DLPEQ-2 approach, the total running time (in one direction) is the time needed for GNT for trying to compute one stable model of the translation TR_1 produced by DLPEQ plus the time needed for GNT for trying to compute one stable model of the translation TR_2 produced by DLPEQ, if there exists no stable models for TR_1. In both approaches the actual translation times are not taken into account, as they are negligible.

In the NAIVE approach, we try to exclude possible overhead due to imple-

menting the NAIVE approach as a shell script. Thus the total running time (in one direction) consists only of the running time for finding the necessary (but not necessarily all) stable models of P plus the individual running times for testing that the stable models found are also stable models of Q.

All the tests were run under the Debian GNU/Linux 2.4.18 operating system on a 450MHz Intel Pentium III computer with 256MB of memory. For the translation-based approaches the reported times are the duration of the search for stable models output by GNT. For the naive approach we measured the "user" time output by the `/usr/bin/time` command in UNIX.

5.4 DISJUNCTIVE RANDOM 3-SAT

We use the Σ_2^{P}-complete problem of finding a minimal model of a SAT *instance* containing specified atoms [5] as our first test problem. A SAT instance in conjunctive normal form (CNF) is a conjunction of clauses $C_1 \wedge C_2 \wedge \cdots \wedge C_c$. A clause is a disjunction of literals $l_1 \vee l_2 \vee \cdots \vee l_n$. Literals are atoms a or their negations $\neg a$. In a SAT instance there are $v = |\{a \mid a \text{ is an atom}\}|$ atoms (also called *variables*) and c clauses. A 3-SAT instance consists of clauses of length three, i.e. all the clauses consist of three literals. A random 3-SAT instance is a 3-SAT instance in which each clause consists of three different literals selected uniformly at random.

We form a disjunctive logic program representing a SAT instance as follows. Each clause $a_1 \vee \cdots \vee a_n \vee \neg b_1 \vee \cdots \vee \neg b_m$ of the instance is translated into a rule $a_1 | \ldots | a_n \leftarrow b_1, \ldots, b_m$. A rule $\bot \leftarrow \sim c_i$ is included for each specified atom c_i. The resulting disjunctive logic program has a stable model if and only if there exists a minimal model for the set of clauses containing all the specified atoms c_i. We generate disjunctive logic programs that each solve an instance of the random 3-SAT problem and add to each program extra rules for random atoms c_i, for $i = 1, \ldots, \lfloor 2v/100 \rfloor$, where v is the number of atoms in the instance.

First we keep the clauses-to-variables ratio c/v constant at 3.5. The disjunctive logic programs generated from such instances typically have several

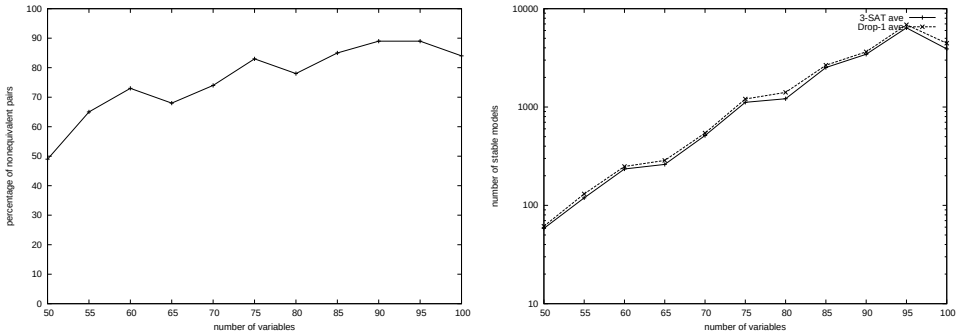

Figure 5.3: Equivalence testing of a disjunctive random 3-SAT instance with constant ratio $c/v = 3.5$ (3-SAT) and its modified version (Drop-1), percentage of nonequivalent pairs and average number of stable models the programs possess.

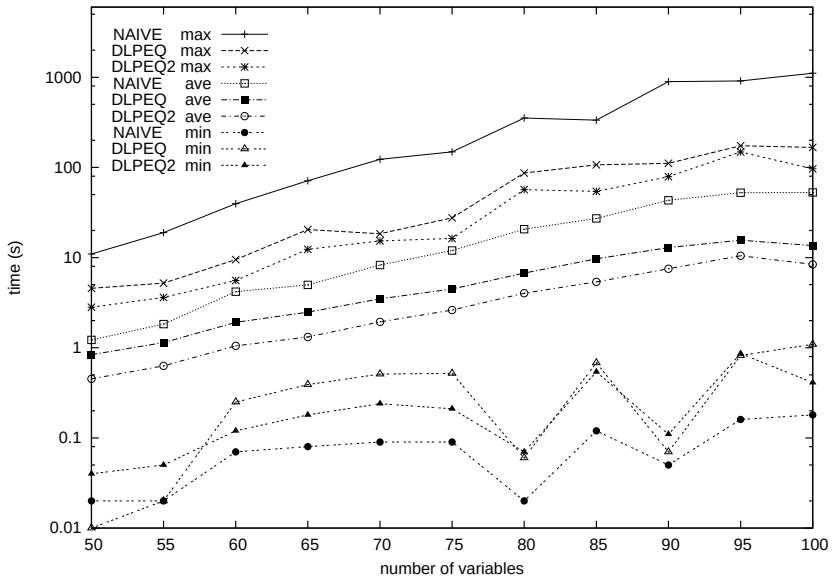

Figure 5.4: Equivalence testing of a disjunctive random 3-SAT instance with constant ratio $c/v = 3.5$ (3-SAT) and its modified version (Drop-1), running times.

stable models (see Figure 5.3). To simulate a sloppy programmer making mistakes, we drop *one* random rule from each program. We test the equivalence of the modified (Drop-1) and the original program to see if making a mistake affects the stable models of the program. Thus the tested pairs of disjunctive logic programs include both equivalent and nonequivalent cases. We vary the number of variables v from 50 to 100 with steps of 5. For each number of variables we repeat the test 100 times generating each time a new random instance. For the problem sizes used, the percentage of nonequivalent cases varies between 60 and 90 (see Figure 5.3). The number of stable models of both the original and the modified program grows exponentially as the problem size grows, with the exception $v = 100$ where the number of stable models drops. At that point a rule for a second specified atom is included to the programs. This reduces the number of stable models.

The maximum, average and minimum running times of the approaches are plotted in Figure 5.4. The DLPEQ approach turns out to be faster than the NAIVE one. The difference in running times increases as instances grow in size. Furthermore, the DLPEQ-2 approach outperforms the DLPEQ approach. Thus, it seems that a translation-based approach really is useful in practice, if the programs to be tested are likely to be nonequivalent and have several stable models. Also, it is worth noticing that the NAIVE approach has outliers (i.e. instances that are exceptionally hard to solve and thus very time-consuming) almost a factor of ten larger than both of the translation-based approaches. On the other hand, the minimum running times of the NAIVE approach are smaller than those of the translation-based approaches.

In the following experiment, we use the same test setting, but keep the clauses-to-variables ratio c/v constant at 3.7. The disjunctive logic programs generated from such instances typically have less stable models as in the previous experiment. The instances are still typically satisfiable. The number

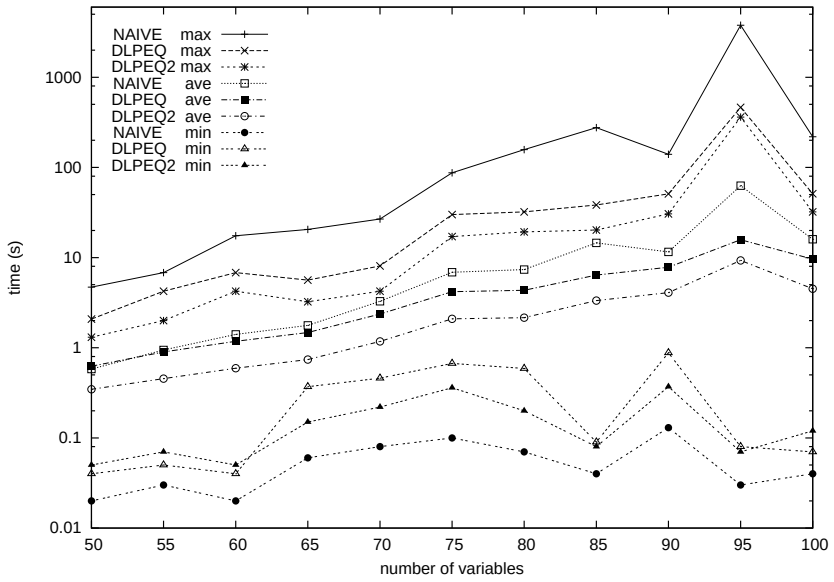

Figure 5.5: Running times for equivalence testing of a disjunctive random 3-SAT instance with constant ratio $c/v = 3.7$ (3-SAT) and its modified version (Drop-1).

of nonequivalent cases is also less than in the previous experiment. For the problem sizes used, the percentage of nonequivalent pairs varies between 35 and 70. It seems that if the programs have less stable models, the dropped rule is less likely to affect the stable models of the program and thus the modified and the original program are more likely equivalent. The maximum, average and minimum running times of all the approaches are plotted in Figure 5.5. The DLPEQ approach is generally slightly faster than the NAIVE one. The DLPEQ-2 approach still outperforms the DLPEQ approach.

In our last experiment with disjunctive 3-SAT, we generate the programs as in the previous experiments, but keep the number of variables constant at $v = 100$, and vary the ratio c/v from 3.25 to 4.50 with steps of 0.25. For each value of the ratio c/v, we repeat the test 100 times, generating each time a new random instance. The motivation is to see how the translation-based approaches perform compared to the NAIVE one as the tested programs change from ones having many stable models to programs having no stable models.

The maximum, average and minimum running times of the approaches are plotted in Figure 5.6. With low values of c/v the DLPEQ and DLPEQ-2 approaches are clearly superior to NAIVE, as in the first experiment. As the ratio increases, the performance of the NAIVE approach gradually improves and finally outperforms the other two approaches. The ratio $c/v \approx 3.75$ is the turning point where the NAIVE approach reaches the DLPEQ approach in performance. At that point, the outliers of the NAIVE approach are still larger than those of the translation-based approaches.

When the ratio c/v is small, the programs to be tested generally have a lot of stable models (see Figure 5.7). The NAIVE approach has to check every stable model in the worst case (i.e. when the logic programs are equivalent) and this takes most of the time the NAIVE approach uses. As the ratio in-

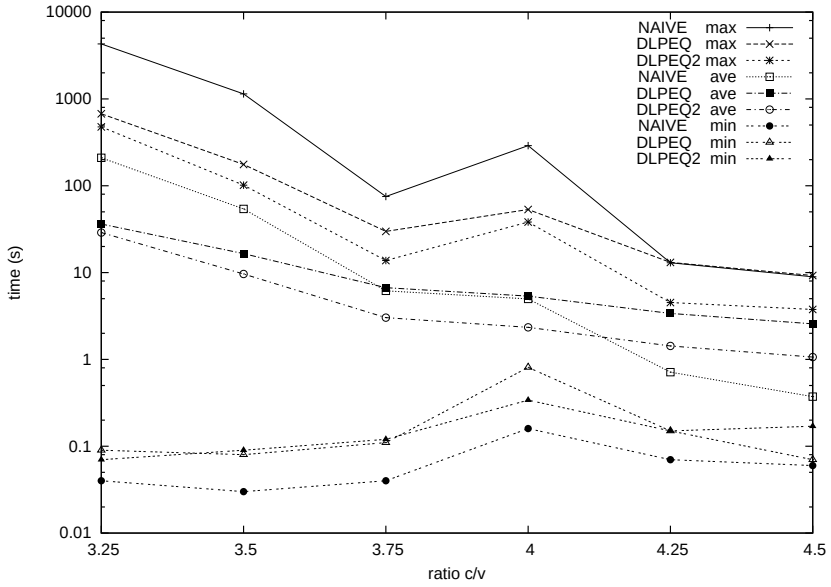

Figure 5.6: Running times for equivalence testing of a disjunctive random 3-SAT instance with constant number of variables $v = 100$ (3-SAT) and its modified version (Drop-1).

creases the programs generally have less and less stable models. Therefore the NAIVE approach improves in performance, as there is need for only few testings of stable models, and the time needed to find the few stable models becomes dominating. The translation-based approaches do not seem to be as sensitive performance-wise to the number of stable models the programs have as the NAIVE approach.

The size of the translation $\mathrm{TR}(P, Q)$ is larger than the sum of the sizes of the original programs. The size of the Herbrand base of the translation is over three times as large as the size of the original Herbrand base. Therefore one might assume that it would be harder to find stable models for the translation. However, based on the three experiments presented in this section, the translation-based approaches seem to be useful in practice, if the

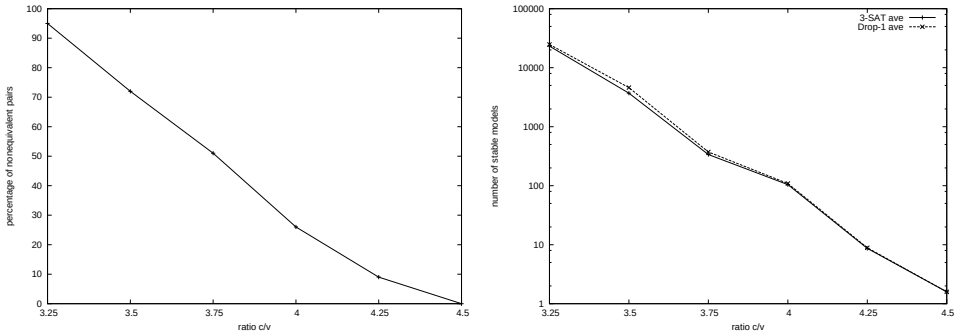

Figure 5.7: Equivalence testing of a disjunctive random 3-SAT instance with constant number of variables, $v = 100$ (3-SAT) and its modified version (Drop-1), percentage of nonequivalent pairs and number of stable models the programs possess.

programs to be tested are likely to have many stable models. We assume that this is due to the fact that the translation provides an exact specification for the counter-example for $P \equiv Q$. This guides the search GNT performs. This is not possible in the NAIVE approach in which the stable models of P and Q are explicitly enumerated. However, if the programs to be tested have few or no stable models, the NAIVE approach is likely to be faster than the DLPEQ approach.

Furthermore, in each experiment the translation $[\mathrm{TR}_1(P,Q), \mathrm{TR}_2(P,Q)]$ is more efficient than the approach based on translation $\mathrm{TR}(P,Q)$. Thus it seems like a good idea to compute the two types of counter-examples separately. Translation $\mathrm{TR}_1(P,Q)$ is compact and efficiently guides the search for counter-examples of type T1. The time needed to test whether the translation $\mathrm{TR}_1(P,Q)$ has stable models is significantly smaller than the time needed to test whether the translation $\mathrm{TR}_2(P,Q)$ has stable models. The more involved translation $\mathrm{TR}_2(P,Q)$ is only used if there is a real need for it. Since the translation for the second phase is only used when there are no counter-examples of type T1, the translation TR_2 is more compact than the one-shot translation TR. Thus, it is generally faster to find a stable model for $\mathrm{TR}_2(P,Q)$ than for $\mathrm{TR}(P,Q)$. Hence, the two-phased translation is as a whole faster and more efficient than the one-shot translation.

5.5 RANDOM 2-QSAT

Our second test problem is QSAT, i.e. the problem of deciding the satisfiability of a *quantified* SAT *formula*. A quantified SAT formula is a SAT instance in which the Boolean variables are either existentially or universally quantified. A k-QSAT problem is a QSAT problem in which there are k alternating quantifiers applied to disjoint sets of variables. The problem QSAT is **PSPACE**-complete, while k-QSAT is $\Sigma_k^{\mathbf{P}}$-complete [32]. We perform our experiments using 2-QSAT instances.

A 2-QSAT-CNF instance is an expression of the form

$$\Phi = \forall X \exists Y \phi,$$

where $X \cap Y = \emptyset$ and ϕ is a SAT formula in conjunctive normal form over the set of variables $X \cup Y$. If $\Phi = \exists X \forall Y \phi$, where ϕ is in disjunctive normal form (i.e. it is a disjunction of conjunctions of literals), we say that Φ is a 2-QSAT-DNF instance.

If ϕ is in conjunctive normal form, then $\neg\phi$ is effectively in disjunctive normal form when the negations are pushed in front of the variables. If ϕ is in conjunctive normal form, we denote $\neg\phi$ that is transformed to disjunctive normal form by $\overline{\phi}$. It holds that

$$\begin{aligned}
&\Phi = \forall X \exists Y \phi \text{ is unsatisfiable} \\
\iff\ &\neg\neg\Phi = \neg\neg(\forall X \exists Y \phi) \text{ is unsatisfiable} \\
\iff\ &\neg\neg\Phi = \neg(\exists X \forall Y \neg\phi) \text{ is unsatisfiable} \\
\iff\ &\neg\Phi = \exists X \forall Y \neg\phi \text{ is valid.}
\end{aligned}$$

We denote by $\overline{\Phi}$ a 2-QSAT-DNF instance $\exists X \forall Y \overline{\phi}$. Thus, if Φ is a 2-QSAT-CNF instance, then Φ is unsatisfiable if and only if $\overline{\Phi}$ is valid.

We define two transformations from 2-QSAT instances to disjunctive logic programs. The first transformation is due to Eiter and Gottlob [5] and it is applied to 2-QSAT-DNF instances.

Definition 5.4. Given a 2-QSAT-DNF instance $\Phi = \exists X \forall Y \phi$, the transformation $\mathrm{EG}(\Phi)$ is a disjunctive logic program containing the following rules where w is a new atom and x' is a new atom introduced for each Boolean variable x in ϕ.

1. a rule $x \mid x'$ for each $x \in X$,
2. rules $y \mid y'$, $y \leftarrow w$ and $y' \leftarrow w$ for each $y \in Y$,
3. a rule $w \leftarrow X_1, X_2', Y_1, Y_2'$ for each disjunct $X_1 \wedge \neg X_2 \wedge Y_1 \wedge \neg Y_2$ in ϕ, where $X_1, X_2 \subseteq X$ and $Y_1, Y_2 \subseteq Y$, and
4. a rule $w \leftarrow \sim w$.

There is a correspondence between the validity of the 2-QSAT-DNF instance Φ and the stable models of the transformation $\mathrm{EG}(\Phi)$ [5].

Theorem 5.5. Φ is a valid 2-QSAT-DNF instance $\iff \mathbf{SM}(\mathrm{EG}(\Phi)) \neq \emptyset$.

For proof of Theorem 5.5 see [5]. The second transformation is due to Janhunen[1] and is applied to 2-QSAT-CNF instances.

Definition 5.6. Given a 2-QSAT-CNF instance $\Phi = \forall X \exists Y \phi$, the transformation $\mathrm{J}(\Phi)$ is a disjunctive logic program containing the following rules where w is a new atom and a new atom x' is introduced for each Boolean variable x in ϕ.

1. rules $x \leftarrow \sim x'$ and $x' \leftarrow \sim x$ for each $x \in X$,
2. rules $y \leftarrow w$ and $y' \leftarrow w$ for each $y \in Y$,
3. a rule $w \mid Y_1 \leftarrow Y_2, \sim X_1, \sim X_2'$ for each clause $X_1 \vee \neg X_2 \vee Y_1 \vee \neg Y_2$ in ϕ, where $X_1, X_2 \subseteq X$ and $Y_1, Y_2 \subseteq Y$,
4. a rule $w \leftarrow \sim w$.

Note that the second rule in the second item in Definition 5.6 is unnecessary and attached to the translation only to expand the Herbrand base of $\mathrm{J}(\Phi)$ to correspond to the Herbrand base of $\mathrm{EG}(\overline{\Phi})$.

Similarly to the transformation $\mathrm{EG}(\cdot)$, there is a correspondence between the validity of the 2-QSAT-CNF instance Φ and the stable models of the transformation $\mathrm{J}(\Phi)$.

Proposition 5.7. Φ is a valid 2-QSAT-CNF instance $\iff \mathbf{SM}(\mathrm{J}(\Phi)) = \emptyset$.

Since $\overline{\Phi}$ is valid if and only if Φ is unsatisfiable, we can use the previous proposition and Theorem 5.5 to conclude that transformations $\mathrm{J}(\Phi)$ and $\mathrm{EG}(\overline{\Phi})$ are equivalent. We will test this proposition experimentally in the following experiments.

Proposition 5.8. $\mathrm{J}(\Phi) \equiv \mathrm{EG}(\overline{\Phi})$, where Φ is a 2-QSAT-CNF instance.

[1] Personal communication, 2003

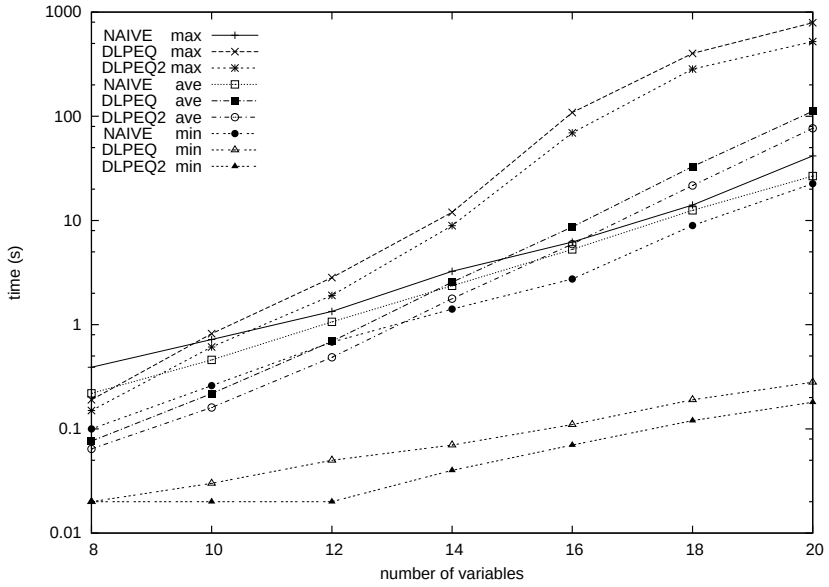

Figure 5.8: Running times for equivalence testing of DLPs $\mathrm{EG}(\overline{\Phi})$ and $\mathrm{J}(\Phi)$, where Φ is a random 2-QSAT-CNF instance with constant ratio $c/v = 3.5$.

In all our experiments with 2-QSAT, $|X| = |Y|$ and $v = |X| + |Y|$. We generate random 2-QSAT-CNF instances $\Phi = \forall X \exists Y \phi$ according to model A presented by Gent and Walsh [12], that is, ϕ is a random 3-SAT instance such that each clause in ϕ contains at least two variables that are existentially quantified. We transform the instances to disjunctive logic programs using transformations presented in Definitions 5.4 and 5.6.

In the first experiment we keep the clauses-to-variables ratio c/v constant at 3.5. We vary the number of variables v from 8 to 24 with steps of 2. We transform the generated instances Φ into disjunctive logic programs $\mathrm{EG}(\overline{\Phi})$ and $\mathrm{J}(\Phi)$ and test the equivalence of the programs obtained. For each number of variables we repeat the test 100 times, generating each time a new random instance.

The results of this experiment support Proposition 5.8, since all the pairs tested are equivalent. The number of stable models of the programs grows exponentially to the number of atoms programs have. The maximum, average and minimum running times of each approach are plotted in Figure 5.8. The translation-based approaches perform better than the naive one when v is small. As the program sizes grow, the NAIVE approach reaches the translation-based approaches in performance and finally outperforms them. Explanation for this is not entirely clear, but we believe that the results are somewhat dependent of the particular problem. It is really easy for GNT to find stable models for the 2-QSAT encodings, and thus the naive approach performs well even though the number of stable models that the programs possess is high. On the other hand, the translations for the equivalence testing are fairly large and hard for GNT to solve.

The running times in the translation-based approaches scale remarkably differently depending on the direction of the translation. It seems that verifying $\mathbf{SM}(\mathrm{TR}(\mathrm{EG}(\overline{\Phi}), \mathrm{J}(\Phi))) = \emptyset$ is easy; most of the total running time is spend verifying that it holds that $\mathbf{SM}(\mathrm{TR}(\mathrm{J}(\Phi), \mathrm{EG}(\overline{\Phi}))) = \emptyset$ (the same ap-

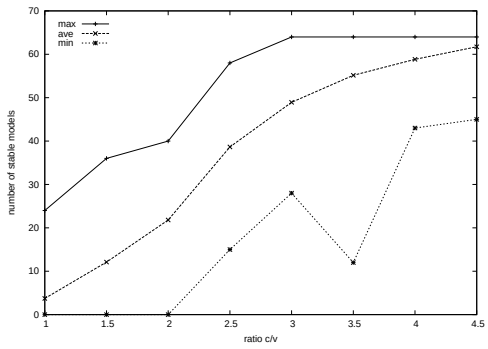

Figure 5.9: Number of stable models of DLPs EG($\overline{\Phi}$) and J(Φ) possess, where Φ is a random 2-QSAT-CNF instance with constant number of variables $v = 12$.

plies for the two-phased approach). In the NAIVE approach there is no such phenomenon to be detected. We believe that this is partly due to the fact that it is somewhat harder for GNT to find stable models for the transformation J(Φ) than for the transformation EG($\overline{\Phi}$). Furthermore, the maximum and minimum running times behave similarly to the experiments with disjunctive random 3-SAT.

Next we keep the number of variables v constant at 12 and vary the clauses-to-variables ratio c/v from 1.0 to 4.5 with steps of 0.5. Similarly to the previous experiment, we transform the generated instances Φ into disjunctive logic programs EG($\overline{\Phi}$) and J(Φ) and test the equivalence of the programs obtained. For each number of variables we repeat the test 100 times, generating each time a new random instance.

When the c/v ratio is small (less than approximately 2) the 2-QSAT-CNF instances Φ are typically satisfiable [12] and thus programs EG($\overline{\Phi}$) and J(Φ) have no or few stable models. As the ratio increases, the 2-QSAT-CNF instances Φ become unsatisfiable and thus the logic program transformations have more stable models. The change in the number of stable models is presented in Figure 5.9. This experiment shows how the different approaches behave when the programs to be tested are equivalent and when the programs change from those having no stable models to those having stable models. It is worth noticing that compared to the similar experiment with disjunctive 3-SAT, the change in the number of stable models is less abrupt.

The maximum, average and minimum running times of each approach are plotted in Figure 5.10. Since the programs are equivalent, the NAIVE approach checks every stable model. Thus when the ratio c/v is small and programs have no stable models, the naive approach is really fast. As the ratio increases, the translation-based approaches improve in performance. In this experiment, the NAIVE approach seems to be less sensitive in its performance than the translation-based ones to the number of stable models the programs possess.

Finally, we transform a generated random instance Φ to a disjunctive logic program EG($\overline{\Phi}$). We test the equivalence of EG($\overline{\Phi}$) and a modified program obtained by dropping *one* random rule from EG($\overline{\Phi}$) (Drop-1). Thus the experiment contains both equivalent and nonequivalent program pairs. We

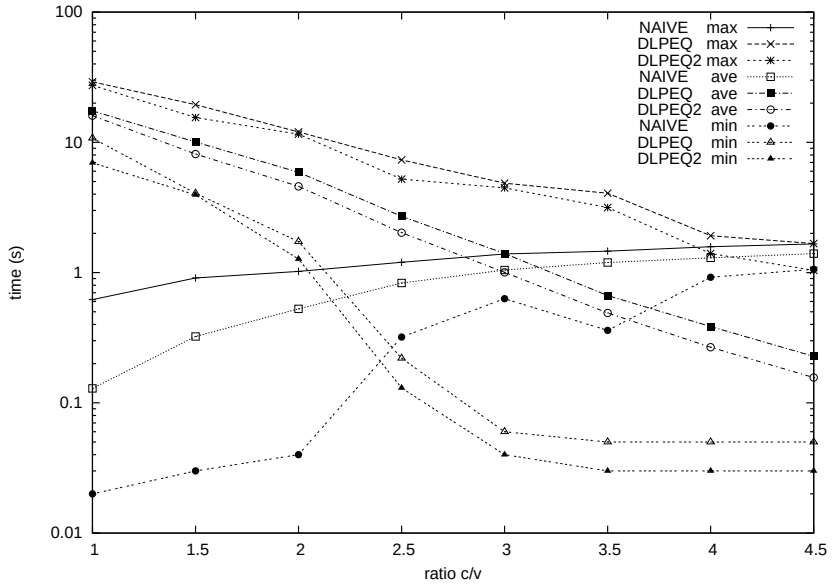

Figure 5.10: Running times for equivalence testing of DLPs EG($\overline{\Phi}$) and J(Φ), where Φ is a random 2-QSAT-CNF instance with constant number of variables, $v = 12$.

keep the clauses-to-variables ratio c/v constant at 3.5 and vary the number of variables v from 10 to 24 with steps of 2. For each number of variables we repeat the test 100 times, generating each time a new random instance. The parameters are selected so that, for the problem sizes used, the percentage of nonequivalent cases is approximately 50–60. The number of stable models of both the original and the modified program grows exponentially as the problem size grows.

The maximum, average and minimum running times of the approaches are plotted in Figure 5.11. The DLPEQ approach is clearly faster than the NAIVE one. The difference in running times increases as instances grow in size. Furthermore, the DLPEQ-2 approach performs slightly better that the DLPEQ approach. Also, it is worth noticing, that in this experiment the minimum running times of the NAIVE approach are over a factor of ten larger than the ones of the translation-based approaches.

Similarly as in the experiment with disjunctive 3-SAT, our experiments with 2-QSAT suggest that if the programs to be tested have few or no stable models, the NAIVE approach is likely to be faster than the DLPEQ approach. The experiments with two equivalent programs also suggest that the NAIVE approach might be a better choice if the programs to be tested are equivalent and have many stable models. This is slightly contradictory to our initial assumptions and requires the use of a more thorough set of experiments to see if the phenomenon is problem-dependent or a more universal one. However, the translation-based approaches are clearly superior to the NAIVE one when the test cases involve equivalent and nonequivalent pairs. Furthermore, in each experiment the DLPEQ-2 approach is faster than the DLPEQ approach. Combined with the experiments with disjunctive random 3-SAT, we can conclude that the two-phased translation is as a whole faster and more efficient than the one-shot translation.

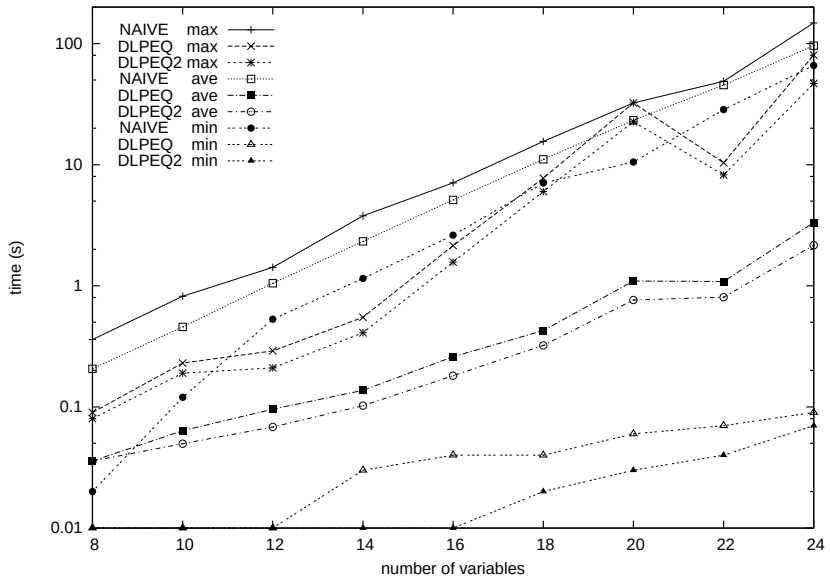

Figure 5.11: Running times for equivalence testing of DLPs EG($\overline{\Phi}$) and Drop-1, where Φ is a random 2-QSAT-CNF instance with constant ratio $c/v = 3.5$.

6 CONCLUSIONS

This work discusses the issues concerning the equivalence testing of disjunctive logic programs under stable model semantics. We establish that verifying the equivalence of disjunctive logic programs is $\Pi_2^{\mathbf{P}}$-complete. Thus, as finding stable models of a disjunctive program is $\Pi_2^{\mathbf{P}}$-complete, there is no complexity theoretical obstacle for extending the translation-based approach for testing the equivalence of logic programs in the case of weight constraint programs [20] to the case of disjunctive programs.

In this work, we present two systematic translations which can be used to reduce the problem of testing whether two logic programs are equivalent to the problem of computing stable models for a disjunctive program. One is a one-shot translation that is used to capture the counter-examples for the equivalence. Thus, to verify the equivalence, one needs to compute the stable models of the translation. As counter-examples for the equivalence divide naturally in two cases (counter-examples of types T1 and T2), the search for counter-examples can easily be divided in two phases. Computational arguments act as the motivation for the other, two-phased, translation. Counter-examples of type T1 can be found using a compact translation, whereas the translation for finding a counter-example of type T2 is more involved. Thus counter-examples of type T2 should be of interest only if counter-examples of type T1 do not exist. The first translation is used to find the counter-examples of type T1 (if such exist) and the second translation is used only if no counter-examples of type T1 exist. The second translation used to find counter-examples of type T2 can also be simplified, since it does not have to capture counter-examples of type T1.

As equivalence testing of disjunctive logic programs can be reduced to

computing stable models of a disjunctive program, existing search engines such as DLV and GNT can be used for the search of counter-examples. There is no need to develop a special purpose system for this task. To enable the equivalence testing of disjunctive programs in practice we have implemented a translator program called DLPEQ. It is compatible with GNT, which can thus be used to compute the stable models of the translation. The current implementation of DLPEQ also enables the use of DLV for finding the stable models, since DLPEQ outputs the translation in a syntax suitable for DLV on demand.

We report experiments with random disjunctive 3-SAT and random 2-QSAT instances. The length of the translation exceeds the sum of the lengths of the programs being tested, and the size of the Herbrand base of the translation is over three times as large as the size of the original Herbrand base. Therefore one could assume that it would be harder to find stable models for the translation than for the original programs. The experiments suggest, however, that the translation-based approach is superior to a naive cross-checking approach when the programs possess many stable models and are likely to be nonequivalent. We assume that this is due to the fact that the translation provides an exact specification for the counter-examples of the equivalence. This guides the search that GNT performs. This is not possible in the NAIVE approach in which the stable models of both programs are explicitly enumerated. However, if the programs to be tested have few or no stable models, the NAIVE approach is likely to be faster than the DLPEQ approach. The experiments with two equivalent 2-QSAT encodings suggest that the NAIVE approach might be faster than the translation-based ones if the programs are equivalent and have a lot of stable models. A further observation is that the two-phased translation $[\mathrm{TR}_1(P,Q), \mathrm{TR}_2(P,Q)]$ is more efficient than the approach based on a one-shot translation $\mathrm{TR}(P,Q)$. Thus it seems like a good idea to compute the two types of counter-examples in isolation.

Based on our experiments we can conclude that there are several cases where the translation-based approaches (especially the two-phased translation) are superior to the naive cross-checking approach. In some cases, however, the use of the naive approach is advisable.

6.1 FUTURE WORK

Finally, there are some improvements and future research to consider.

- We assume that $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ holds for programs P and Q under comparison. If $\mathrm{Hb}(P) \neq \mathrm{Hb}(Q)$, then one solution is to add useless rules (for example of the form $a \leftarrow a$) to P and Q such that their stable models are not affected and $\mathrm{Hb}(P') = \mathrm{Hb}(Q')$ holds for the resulting programs P' and Q'.

- All the experiments presented in this work evaluating the feasibility of the translation-based approaches involve random instances. The performance of the translation functions should be examined using a more thorough set of experiments involving program instances modeling real-life problems that lie on the second level of the polynomial hi-

erarchy. One such real-life example is the problem of verifying whether different concepts of diagnosis [3, 4, 26], applicable e.g. to digital circuits, are equivalent.

- Also, in all the experiments presented we use the solver GNT. We decided to use only one solver, since at this stage we wanted to obtain information how the the performance of the translation-based approach performs compared to the NAIVE approach. However, is not yet clear how the choice of a solver affects the performance of the approaches. Thus a further set of experiments where different solvers are used to find the stable models would offer more information on the overall efficiency of the translation-based approach. We believe that the results would be very similar to the ones obtained in our present experiments if another solver was used.

- Equivalence testing in even more general classes of logic programs can be covered through suitable translations. For instance, the equivalence of *nested programs* [22] can be covered by implementing the transformation of nested programs into disjunctive programs [33] and using the translation presented in this work for the resulting disjunctive programs.

- Translation-based approach to equivalence testing could also be extended to cover the notion of strong equivalence. Testing the strong equivalence of nested (as well as disjunctive) programs is computationally less complex than testing the weak equivalence, which suggests using an **NP** solver such as SMODELS for the strong case instead of GNT or DLV. The current implementation of LPEQ supports strong equivalence testing of normal logic programs. Thus strong equivalence testing of disjunctive programs is not yet covered. Also, Lin [24] gives a transformation that enables the use of SAT solvers for testing strong equivalence.

ACKNOWLEDGEMENTS

This is a report version of my Master's thesis [31]. The thesis is result of studies and research in Laboratory for Theoretical Computer Science at Helsinki University of Technology. The research has been funded by Academy of Finland (project 53695). I am very grateful to my instructor Docent Tomi Janhunen for his advice and valuable comments. I would like to thank my supervisor and the head of the laboratory Professor Ilkka Niemelä for his comments on the thesis and for giving me the opportunity to concentrate to this work. I would also like to thank my colleagues in the laboratory for creating a pleasant working atmosphere.

I thank warmly my parents for the love, support and encouragement they have always given me. I thank my sister and brother as well as all my friends for being there for me and for dragging me out to the real world from time to time. Finally, I thank my colleague and dearest friend Matti Järvisalo. Without his friendship and support my life would have been much harder during this process.

BIBLIOGRAPHY

[1] Stefan Brass and Jürgen Dix. A characterization of the stable semantics by partial evaluation. In Victor W. Marek, Anil Nerode, and Miroslaw Truszczyński, editors, *Logic Programming and Nonmonotonic Reasoning: Proceedings of the Third International Conference*, pages 85–98, Berlin, Germany, June 1995. Springer. LNAI 928.

[2] Keith L. Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[3] Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing diagnosis and systems. *Artificial Intelligence*, 56(2–3):54–65, 1992.

[4] Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *Journal of the ACM*, 42(1):3–42, 1995.

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

[6] Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR system DLV: Progress report, comparisons and benchmarks. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, pages 406–417, Trento, Italy, June 1998. Morgan Kaufmann.

[7] Esra Erdem, Vladimir Lifschitz, and Martin Wong. Wire routing and satisfiability planning. In John Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Luís Moniz Pereira, Yehoshua Sagiv, and Peter Stuckey, editors, *Proceedings of the First International Conference on Computational Logic, Automated Deduction: Putting Theory into Practise*, pages 822–836, London, UK, July 2000. Springer-Verlag. LNCS 1861.

[8] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. Available at `http://www.cs.utexas.edu/users/vl/papers.html`, 2003. Unpublished draft.

[9] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert A. Kowalski and Kenneth Bowen, editors, *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, Seattle, USA, August 1988. The MIT Press.

[10] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In David H. D. Warren and Peter Szeredi, editors, *Proceedings of the 7th International Conference on Logic Programming*, pages 579–597, Jerusalem, Israel, June 1990. The MIT Press.

[11] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9(3–4):365–385, 1991.

[12] Ian Gent and Toby Walsh. Beyond NP: The QSAT phase transition. In James Hendler and Devika Subramanian, editors, *AAAI: 16th National Conference on Artificial Intelligence*, pages 648 – 653, Orlando, Florida, USA, July 1999. The MIT Press.

[13] Keijo Heljanko. Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe petri nets. *Fundamenta Informaticae*, 37(3):247–268, 1999.

[14] Maarit Hietalahti, Fabio Massacci, and Ilkka Niemelä. DES: a challenge problem for non-monotonic reasoning systems. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*, Colorado, USA, April 2000. CoRR:cs.AI/0003039.

[15] Katsumi Inoue and Chiaki Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35(1):39–78, 1998.

[16] Tomi Janhunen. On the effect of default negation on the expressiveness of disjunctive rules. In Thomas Eiter, Wolfgang Faber, and Miroslaw Truszczyński, editors, *Logic Programming and Nonmonotonic Reasoning, Proceedings of the 6th International Conference*, pages 93–106, Vienna, Austria, September 2001. Springer-Verlag. LNAI 2173.

[17] Tomi Janhunen. LPEQ 1.13 — a tool for testing the equivalence of logic programs. `http://www.tcs.hut.fi/Software/lpeq/`, 2002. Computer Program.

[18] Tomi Janhunen. GnT 2 — a tool for computing disjunctive stable models. `http://www.tcs.hut.fi/Software/gnt/`, 2003. Computer Program.

[19] Tomi Janhunen, Ilkka Niemelä, Dietmar Seipel, Patrik Simons, and Jia-Huai You. Unfolding partiality and disjunctions in stable model semantics. CoRR: cs.AI/0303009, March 2003. Submitted for publication.

[20] Tomi Janhunen and Emilia Oikarinen. Testing the equivalence of logic programs under stable model semantics. In Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni, editors, *Logics in Artificial Intelligence, Proceedings of the 8th European Conference*, pages 493–504, Cosenza, Italy, September 2002. Springer-Verlag. LNAI 2424.

[21] Vladimir Lifschitz. Answer set planning. In Danny de Schreye, editor, *Proceedings of the 16th International Conference on Logic Programming*, pages 23–37, Las Cruces, New Mexico, USA, December 1999. The MIT Press.

[22] Vladimir Lifschitz, David Pearce, and Augustin Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2(4):526–541, 2001.

[23] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics in Artificial Intelligence*, 25(3–4):369–389, 1999.

[24] Fangzhen Lin. Reducing strong equivalence of logic programs to entailment in classical propositional logic. In Dieter Fensel, Fausto Giunchiglia, Deborah McGuinness, and Williams Mary-Anne, editors, *Principles of Knowledge Representation and Reasoning: Proceedings of the 8th International Conference*, pages 170–176, Toulouse, France, April 2002. Morgan Kaufmann.

[25] John Wylie Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1987.

[26] Peter Lucas. Analysis of notions of diagnosis. *Artificial Intelligence*, 105(1–2):295–343, 1998.

[27] Viktor W. Marek and Miroslaw Truszczyński. Autoepistemic logic. *Journal of the ACM*, 38(3):588–619, 1991.

[28] Viktor W. Marek and Miroslaw Truszczyński. Stable models and an alternative logic programming paradigm. In Krzysztof R. Apt, Viktor W. Marek, Miroslav Truszczyński, and David S. Warren, editors, *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.

[29] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3–4):241–273, 1999.

[30] Emilia Oikarinen. DLPEQ 1.7 — a tool for testing the equivalence of disjunctive logic programs. `http://www.tcs.hut.fi/Software/lpeq/`, 2003. Computer Program.

[31] Emilia Oikarinen. Testing the equivalence of disjunctive logic programs. Master's thesis, Helsinki University of Technology, Department of Computer Science and Engineering, Laboratory for Theoretical Computer Science, Espoo, Finland, September 2003.

[32] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, Reading, Massachusetts, USA, 1994.

[33] David Pearce, Vladimir Sarsakov, Torsten Schaub, Hans Tompits, and Stefan Woltran. A polynomial translation of logic programs with nested expressions into disjunctive logic programs: Preliminary report. In Peter J. Stuckey, editor, *Proceedings of the 18th International Conference on Logic Programming*, pages 405–420, Copenhagen, Denmark, July–August 2002. Springer-Verlag. LNCS 2401.

[34] David Pearce, Hans Tompits, and Stefan Woltran. Encodings for equilibrium logic and logic programs with nested expressions. In Pavel Brazdil and Jorge Alípio, editors, *Proceedings of the 10th Portuguese Conference on AI, EPIA'01*, pages 306–320, Porto, Portugal, December 2001. Springer-Verlag. LNAI 2258.

[35] Teodor Przymusinski. Extended stable semantics for normal and disjunctive logic programs. In David Warren and Peter Szeredi, editors, *Proceedings of the 7th International Conference on Logic Programming*, pages 459–477, Jerusalem, Israel, June 1990. The MIT Press.

[36] Teodor Przymusinski. Stable semantics for disjunctive logic programs. *New generation Computing*, 9(3–4):401–424, 1991.

[37] Patrik Simons, Ilkka Niemelä, and Timo Soininen. Extending and implementing the stable model semantics. *Artificial Intelligence*, 138(1–2):181–234, 2002.

[38] Timo Soininen and Ilkka Niemelä. Developing a declarative rule language for applications in product configuration. In Gopal Gupta, editor, *Proceedings of the First International Workshop on Practical Aspects of Declarative Languages*, pages 305–319, San Antonio, Texas, USA, January 1999. Springer-Verlag.

[39] L. Stockmeyer. The polynomial hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[40] Hudson Turner. Strong equivalence made easy: Nested expressions and weight constraints. *Theory and Practice of Logic Programming*, 3(4–5):609–622, 2003.

HELSINKI UNIVERSITY OF TECHNOLOGY LABORATORY FOR THEORETICAL COMPUTER SCIENCE
RESEARCH REPORTS

HUT-TCS-A72 Tommi Junttila
Symmetry Reduction Algorithms for Data Symmetries. May 2002.

HUT-TCS-A73 Toni Jussila
Bounded Model Checking for Verifying Concurrent Programs. August 2002.

HUT-TCS-A74 Sam Sandqvist
Aspects of Modelling and Simulation of Genetic Algorithms: A Formal Approach.
September 2002.

HUT-TCS-A75 Tommi Junttila
New Canonical Representative Marking Algorithms for Place/Transition-Nets. October 2002.

HUT-TCS-A76 Timo Latvala
On Model Checking Safety Properties. December 2002.

HUT-TCS-A77 Satu Virtanen
Properties of Nonuniform Random Graph Models. May 2003.

HUT-TCS-A78 Petteri Kaski
A Census of Steiner Triple Systems and Some Related Combinatorial Objects. June 2003.

HUT-TCS-A79 Heikki Tauriainen
Nested Emptiness Search for Generalized Büchi Automata. July 2003.

HUT-TCS-A80 Tommi Junttila
On the Symmetry Reduction Method for Petri Nets and Similar Formalisms.
September 2003.

HUT-TCS-A81 Marko Mäkelä
Efficient Computer-Aided Verification of Parallel and Distributed Software Systems.
November 2003.

HUT-TCS-A82 Tomi Janhunen
Translatability and Intranslatability Results for Certain Classes of Logic Programs.
November 2003.

HUT-TCS-A83 Heikki Tauriainen
On Translating Linear Temporal Logic into Alternating and Nondeterministic Automata.
December 2003.

HUT-TCS-A84 Johan Wallén
On the Differential and Linear Properties of Addition. December 2003.

HUT-TCS-A85 Emilia Oikarinen
Testing the Equivalence of Disjunctive Logic Programs. December 2003.

ISBN 951-22-6878-7
ISSN 1457-7615