Helsinki University of Technology Laboratory for Theoretical Computer Science
Research Reports 106
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 106
Espoo 2006 HUT-TCS-A106

# MODULAR ANSWER SET PROGRAMMING

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 106
Teknillisen korkeakoulun tietojenkäsittelyteorian laboratorion tutkimusraportti 106
Espoo 2006 HUT-TCS-A106

# MODULAR ANSWER SET PROGRAMMING

**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
FI-02015 TKK, FINLAND
Tel. +358 9 451 1
Fax. +358 9 451 3369
E-mail: lab@tcs.tkk.fi
URL: http://www.tcs.tkk.fi/

ISBN-13 978-951-22-8581-5
ISBN-10 951-22-8581-9
ISSN 1457-7615

Multiprint Oy
Helsinki 2006

**Abstract:** Answer set programming (ASP) is a declarative rule-based constraint programming paradigm. In ASP the problem at hand is solved declaratively by writing down a logic program the answer sets of which correspond to the solutions of the problem, and then computing the answer sets of the program using a special purpose search engine. The growing interest towards ASP is mostly due to efficient search engines available today. Consequently, a variety of interesting applications of ASP has emerged, for example, in planning, product configuration, computer aided verification, and wire routing in VLSI design.

Despite the declarative nature of ASP the development of programs resembles that of programs in conventional programming: a programmer often develops a series of gradually improving programs for a particular problem, for example, when optimizing execution time and space. Currently ASP programs are considered as integral entities. This becomes problematic as programs become more complex, and the sizes of program instances grow. In ASP there is a lack of mechanisms, available in other modern programming languages, that ease program development by allowing re-use of code or breaking programs into smaller pieces, modules. Even though modularity has been studied extensively in conventional logic programming, there are only few approaches how to incorporate modularity into ASP.

In this report we propose a simple and intuitive notion of a logic program module that interacts through an input/output interface. The module system is fully compatible with the stable model semantics. This is achieved by restricting the composition of modules in a way that module-level stability implies program-level stability, and vice versa. Furthermore, we introduce a notion of modular equivalence that is a proper congruence relation for the composition of modules and analyze the computational complexity of deciding modular equivalence. We extend an earlier translation-based method for verifying equivalence of ASP programs to cover the verification of modular equivalence of SMODELS program modules, and evaluate experimentally the efficiency of the translation-based method in the verification of modular equivalence. We also study questions related to finding a suitable module structure for a program when there is no explicit a priori knowledge on the underlying structure.

**Keywords:** modular answer set programming, equivalence verification, modular congruence, stable model semantics, nonmonotonic reasoning

# CONTENTS

# LIST OF ABBREVIATIONS AND NOTATIONS

| | | |
|---|---|---|
| ASP | answer set programming | 1 |
| NLP | normal logic program | 8 |
| SCC | strongly connected component | 10 |
| EVA | property of having enough visible atoms | 14 |
| | | |
| SMODELS | a state-of-the-art answer set solver | 40 |
| LPARSE | front-end of of SMODELS | 40 |
| LPEQ | tool for verifying equivalence of logic programs | 40 |
| LPCAT | tool for joining logic programs/modules | 40 |
| | | |
| $\sim$ | negation as failure | 8 |
| $P, Q, R, \ldots$ | logic programs | 8 |
| $\mathrm{Head}(\cdot)$ | set of head atoms in set of rules | 8 |
| $B^{+}, \mathrm{Body}^{+}(\cdot)$ | set of positive body atoms in a rule | 8 |
| $B^{-}, \mathrm{Body}^{-}(\cdot)$ | set of negative body atoms in a rule | 8 |
| $\mathrm{CompS}(\cdot)$ | set of literals in compute statements | 27 |
| $\mathrm{WS}_{M}(\cdot)$ | sum of weights of literals satisfied in $M$ | 27 |
| | | |
| $\mathrm{Hb}(\cdot)$ | Herbrand base of a program | 8 |
| $\mathrm{Hb}_{\mathrm{v}}(\cdot)$ | visible Herbrand base of a program | 9 |
| $\mathrm{Hb}_{\mathrm{h}}(\cdot)$ | hidden Herbrand base of a program | 9 |
| $\mathrm{atoms}_{\mathrm{v}}(\cdot)$ | set of visible atoms appearing in a set of rules | 38 |
| | | |
| $A_{\mathrm{v}}$ | visible part of set of atoms $A$ | 14 |
| $A_{\mathrm{h}}$ | hidden part of set of atoms $A$ | 14 |
| $A_{\mathrm{i}}$ | input part of set of atoms $A$ | 33 |
| $A_{\mathrm{o}}$ | output part of set of atoms $A$ | 33 |
| | | |
| $M, N, \ldots$ | interpretations | 9 |
| $\models$ | (classical) satisfaction relation | 9 |
| $P^{M}$ | Gelfond-Lifschitz reduct of $P$ with respect to $M$ | 9 |
| $\mathrm{LM}(\cdot)$ | least model of a program | 9 |
| $\mathrm{SM}(\cdot)$ | set of stable models for a program | 9 |
| | | |
| $U$ | splitting set | 10 |
| $\mathrm{b}_{U}(P)$ | bottom of $P$ relative to splitting set $U$ | 10 |
| $\mathrm{t}_{U}(P)$ | top of $P$ relative to splitting set $U$ | 10 |
| $\mathrm{e}(P, X)$ | partial evaluation of $P$ with respect to $X$ | 10 |
| $\langle X, Y\rangle$ | solution to a program | 10 |
| | | |
| $\mathrm{Dep}^{+}(\cdot)$ | positive dependency graph of a program | 10 |
| $\mathrm{Dep}^{\mathrm{h}}(\cdot,\cdot)$ | hidden dependency graph of a set of SCCs | 38 |

| | | |
|---|---|---|
| $\mathrm{Def}_P(a)$ | the set of rules in program $P$ defining atom $a$ | 11 |
| $P[C]$ | set of rules in $P$ defining set of atoms $C$ | 11 |
| | | |
| $\equiv$ | weak/ordinary equivalence | 11 |
| $\equiv_{\mathrm{s}}$ | strong equivalence | 11 |
| $\equiv_{\mathrm{u}}$ | uniform equivalence | 12 |
| $\equiv_{\mathrm{s}}^{A}$ | $A$-strong equivalence | 13 |
| $\equiv_{\mathrm{u}}^{A}$ | $A$-uniform equivalence | 13 |
| $\equiv_{\mathrm{v}}$ | visible equivalence | 13 |
| $\equiv_{\mathrm{w}}$ | weak visible equivalence | 15 |
| $\equiv_{\mathrm{m}}$ | modular equivalence | 23 |
| $\equiv_{\mathrm{m}}^{\mathrm{s}}$ | strong modular equivalence | 24 |
| | | |
| $P_{\mathrm{h}}/M_{\mathrm{v}}$ | hidden part of $P$ relative to $M_{\mathrm{v}}$ | 14 |
| $\mathrm{simp}(P,T,F)$ | simplification of $P$ with respect to $T$ and $F$ | 14 |
| | | |
| $\mathbb{P},\mathbb{Q},\mathbb{R}\ldots$ | logic program modules | 16 |
| $\mathbb{P}=(P,I,O)$ | module with set of rules $P$, input $I$ and output $O$ | 16 |
| $\sqcup$ | composition operator for modules | 17 |
| $\mathrm{F}_A$ | logic program defining a set of facts $A$ | 19 |
| $\mathbb{P}(A)$ | $\mathbb{P}$ instantiated with an actual input $A\subseteq I$ | 18 |
| $\mathbb{P}[C]$ | submodule of $\mathbb{P}$ induced by set of atoms $C$ | 38 |
| | | |
| $\mathrm{EQT}(\cdot,\cdot)$ | translation for verifying modular equivalence | 32 |
| $\mathrm{Hidden}^{\circ}(\cdot)$ | submodule of translation $\mathrm{EQT}(\cdot,\cdot)$ | 33 |
| $\mathrm{Least}^{\bullet}(\cdot)$ | submodule of translation $\mathrm{EQT}(\cdot,\cdot)$ | 33 |
| $\mathrm{UnStable}(\cdot)$ | submodule of translation $\mathrm{EQT}(\cdot,\cdot)$ | 34 |

# LIST OF FIGURES

# 1 INTRODUCTION

Answer set programming [54, 50, 19] is an approach to declarative rule-based constraint programming that has received increasing attention over the last few years. In answer set programming (ASP) the problem at hand is solved declaratively by writing down a logic program the answer sets of which correspond to the solutions of the problem, and then computing the answer sets of the program using a special purpose search engine. The growing interest towards answer set programming is mostly due to efficient search engines, such as SMODELS [65], DLV [33], GNT [29], ASSAT [41], CMODELS-2 [34], PBMODELS [44], NOMORE++ [1], CLASP [18], and SAG [42] available today. Consequently, a variety of interesting applications of ASP has emerged, for example, in planning [35], product configuration [66], computer aided verification [24], wire routing in VLSI design [13], logical cryptanalysis [25], and a decision support system of NASA space shuttle [2].

Despite the declarative nature of ASP the development of programs resembles that of programs in conventional programming, that is, a programmer often develops a series of gradually improving programs for a particular problem, for example, when optimizing execution time and space. The development and optimization of programs in ASP gives rise to a meta-level problem of verifying whether subsequent programs are equivalent. There are several notions of equivalence proposed for logic programs. For instance, if logic programs $P$ and $Q$ have exactly the same answer sets, they are said to be *weakly/ordinarily equivalent*, denoted by $P \equiv Q$. Looking this from the answer set programming perspective, weakly equivalent programs produce the same solutions for the problem they formalize. If $P \cup R \equiv Q \cup R$ for all programs $R$, then $P$ and $Q$ are said to be *strongly equivalent*, denoted by $P \equiv_{\mathrm{s}} Q$. Strongly equivalent programs preserve the solutions to the problem in every possible context in which they can be placed in.

A translation-based approach has been proposed and extended further for solving the equivalence verification problem, see for instance, [30, 69, 56, 71, 31]. The underlying idea is to combine logic programs $P$ and $Q$ under consideration into logic programs $\mathrm{EQT}(P, Q)$ and $\mathrm{EQT}(Q, P)$ which have no answer sets if and only if $P$ and $Q$ are equivalent. This enables the use of the same ASP solver, such as SMODELS or DLV, for the equivalence verification task as for the search of answer sets in general. Note, however, that programs are treated as integral entities in the translation-based method. This might limit the usefulness of the translation-based method, for example, in a situation where there is a small local change in a large program. It seems likely that one could seek computational advantage by breaking programs into smaller pieces, that is, *modules*, and by verifying equivalence of modules instead of complete programs.

The same line of thinking applies to current ASP methodology in general. A program in answer set programming is considered as an integral entity, and there is a lack of mechanisms available in modern programming languages that ease program development by allowing re-use of code or breaking programs into smaller pieces. This becomes problematic when programs become more complex, and the sizes of program instances grow. Furthermore, current ASP tools require users to have rather extensive knowledge on ASP methodology. We want to make program development in answer set programming easier and, more generally, make ASP methodology more accessible for specialists in other fields than computer science. Modularization of answer set programming is a way to structure and ease the program development process, and this way answer set programming can become an even more attractive approach for solving hard combinatorial problems, for example, in the areas of semantic web, bioinformatics, and cryptology.

The aim of this report is to develop answer set programming into a more module-oriented direction in which ASP programs consist of modules that interact through suitable interfaces. Program optimization would thus involve module-level optimization and, for example, a suitable *equivalence relation* is needed to justify the replacement of a module with another, that is, to be able to guarantee that changes made on the level of modules do not alter the models of the program when seen as an entity.

The rest of this chapter is organized as follows. We start by related work in Section 1.1, and list the design criteria and goals for the module system in Section 1.2. The contributions of this report are presented in Section 1.3.

## 1.1 RELATED WORK

Modularity has been studied extensively in *conventional logic programming*, see the survey by Bugliesi et al. [5], but there are only few approaches how to incorporate *modularity into answer set programming* [11, 26, 15, 68]. In this section we review some of the approaches proposed.

### 1.1.1 Modularity in Conventional Logic Programming

Our discussion of approaches to modularity in conventional logic programming is mainly based on the extensive survey by Bugliesi et al. [5]. When considering what is needed from a modular logic programming language, several properties can be highlighted [5], for instance, modular language should

- *allow abstraction, parameterization, and information hiding,*
- *ease program development and maintenance of large programs,*
- *allow re-usability,*
- have a *non-trivial notion of program equivalence* to justify replacement of program components, and
- maintain the *declarativity of logic programming.*

Bugliesi et al. [5] identify two mainstream programming disciplines: *programming-in-the-large* where programs are composed with algebraic operators (for instance [4, 17, 48, 59]) and *programming-in-the-small* with abstraction mechanisms (for instance [23, 52]).

The *programming-in-the-large* approaches have their roots in O'Keefe's work [59] where logic programs are seen as an *elements of an algebra* and the operators for *composing programs* are seen as *operators in that algebra*. The fundamental idea is that a logic program should be understood as a part of a system of programs. Program composition is a powerful tool for structuring programs without any need to extend the underlying language of Horn clauses. Several algebraic operations such as *union*, *deletion*, *overriding union* and *closure* have been considered. This approach *supports* naturally the *re-use* of the pieces of programs in different composite programs, and when combined with an adequate equivalence relation also the replacement of equivalent components. This approach is highly flexible, as new composition mechanisms can be obtained by introducing a corresponding operator in the algebra or combining existing ones. *Encapsulation and information hiding* can be obtained by introducing suitable *interfaces* between components. For example, Mancarella and Pedreschi [48], Brogi et al. [4], and Gaifman and Shapiro [17] present compositional frameworks that can be seen as different formulations of O'Keefe's ideas.

The *programming-in-the-small* approaches originate from Miller's work [52]. In his approach the composition of modules is modelled in terms of logical connectives of a language that is defined as an *extension of Horn clause logic*. Giordano and Martelli's approach [23] employs the same structural properties, but suggests a more refined way of modelling visibility rules than the one given in [52]. We focus in the programming-in-the-large approaches in more detail, since the syntax of logic programs in answer set programming is already more general than that of *Horn logic programs* used in conventional logic programming. In addition, for example, *aggregates* can be used as abstraction mechanisms in ASP.

## 1.1.2 Compositionality and Full Abstraction

Maher [47] states that the very least to be expected from a semantical characterization of a modular language is that the meaning of composite programs can be defined in terms of the meaning of its components. To be able to identify when it is safe to substitute two modules with one another without effecting the global behaviour it is crucial to have a notion of *semantical equivalence*. More formally [17, 51] these desired properties can be described under the terms of *compositionality* and *full abstraction*. Two programs are *observationally congruent*, if and only if they exhibit the same observational behaviour in every *context* they can be placed in. A semantics is compositional if semantical equality implies observational congruence. Full abstraction means then that semantical equivalence coincides with observational congruence.

Maher [46] studies compositionality and full abstraction properties for different notions of semantical equivalence (*subsumption equivalence*, *logical equivalence*, and *minimal Herbrand model equivalence*) and different operators in an algebra (union, closure, overriding union). It is worth noting that minimal Herbrand model equivalence coincides with the weak equivalence relation $\equiv$ for positive logic programs. Maher [46] shows that the equivalence based on minimal Herbrand model semantics is not compositional with respect to union. Therefore it is clear that for our purposes union is not suitable composition operator as such, and some restrictions are needed. Also, we see that closure and deletion do not necessarily have a suitable meaning in answer set programming, as we are interested in models, not queries. For instance, the closure property is already inbuilt in answer set programming.

### 1.1.3 Approach by Gaifman and Shapiro

As an example of a programming-in-the-large approach that is *compositional* and *fully abstract* we look at more detail the framework proposed by Gaifman and Shapiro [17]. They consider the language of *definite logic programs*, that is, clauses of the form $A \leftarrow B_1, \ldots, B_n$, where $A$, $B_1$,…, $B_n$ are atoms. Atoms are predicates instantiated with *terms*, and can thus contain *function symbols* and *variables* in addition to constants. The semantics considered is based on *atomic consequences*[1], that is, an atom $A$ is a logical consequence of a program $P$ if and only if $A$ is derivable from $P$ via SLD resolutions.

A *logic module* $L$ is a set of clauses with partitioning of predicates into *imported, exported* and *internal* ones, that is, $L$ is a quadruple

$$L = (P, Im, Ex, Int).$$

An *imported predicate* is supplied to the module by the environment, for example, another module, and it *cannot appear in the head of a clause*. Other predicates can appear anywhere. *External predicates can be supplied to other modules*, while *internal predicates cannot*. *Communication between modules* is achieved through *predicate sharing*. Two modules $L_1 = (P_1, Im_1, Ex_1, Int_1)$ and $L_2 = (P_2, Im_2, Ex_2, Int_2)$ are *composable*, if $Int_1$ and $Int_2$ are local to $L_1$ and $L_2$ and $Ex_1 \cap Ex_2 = \emptyset$. Composition of modules is defined as

$$L_1 + L_2 = (P_1 \cup P_2, (Im_1 \cup Im_2) \setminus (Ex_1 \cup Ex_2), Ex_1 \cup Ex_2, Int_1 \cup Int_2).$$

Semantics for modules is defined by taking into account the interface restrictions. A clause $A \leftarrow B_1, \ldots, B_n$ is an *Import/Export clause* (I/E-clause) of $L$, if $A \in Ex$ and $\{B_1, \ldots, B_n\} \subseteq Im$. An *I/E consequence* of $L$ is a logical consequence of $L$ which is an I/E clause, and an *atomic I/E consequence* is an *atomic* consequence whose predicate is exported. Observational congruence with respect to composition $+$ is defined in terms of *atomic I/E consequences*, that is,

$$Ob_{GS}(L) = \{a \mid A \text{ is an atomic I/E consequence of } L\}.$$

Semantical equivalence is defined as follows:

> Modules $L_1$ and $L_2$ are semantically equivalent if and only if $L_1$ and $L_2$ have the same *minimal* [2] *I/E consequences*.

The system is now *compositional* and *fully abstract* as atomic I/E consequences are exactly the minimal I/E consequences [17, Theorem 11].

[1]This is different from answer set programming where the semantics is based on models. Note, however, that least models considered in answer set programming coincide with atomic consequences if one considers propositional (variable-free) positive logic programs.

[2]A clause $C$ is minimal in $V$ if it is not tautological, there is no proper subclause of $C$ in $V$ and $C$ is not an instance of another $C' \in V$ with the same number of literals. An I/E consequence is minimal if it is minimal in the set of I/E consequences.

### 1.1.4 Modularity in Answer Set Programming

There are a number of approaches within answer set programming involving modularity in some sense, but only few of them really describe a flexible module architecture with a clearly defined interface for module interaction.

The approach of Eiter, Gottlob, and Veith [11] addresses modularity in answer set programming in the programming-in-the-small sense. They view program modules as *generalized quantifiers* [43, 53] the definitions of which are allowed to nest, that is, program $P$ can refer to another module $Q$ by using it as a generalized quantifier. The main program is clearly distinguished from subprograms, and it is possible to nest calls to submodules if the so-called *call graph* is *hierarchical*, that is, *acyclic*. Nesting, however, raises the computational complexity depending on the depth of nesting. Ianni et al. [26] have another programming-in-the-small approach for modularity in ASP based on *templates*.

Tari, Baral, and Anwar [68] extend the language of normal logic programs by introducing the concept of *import rules* for their ASP program modules. There are three types of import rules:

(a) $q(\overline{X}) \leftarrow M(\bar{b}).p(\overline{X}, \overline{a})$,

(b) $*q(\overline{X}) \leftarrow M(\bar{b}).p(\overline{X}, \overline{a})$, and

(c) $q(\#, \overline{X}) \leftarrow M(\bar{b}).p(\overline{X}, \overline{a})$,

where $p$ and $q$ are predicate names, $M$ is a module name, $\overline{X}$ a tuple of variables and $\overline{a}$ and $\bar{b}$ tuples of constants. Import rules are used to import set of tuples $\overline{X}$ for $q$ from module $M(\bar{b})$. Rule (a) imports tuples $\overline{X}$ such that $p(\overline{X}, \overline{a})$ is true in *all answer sets* of $M(\bar{b})$. Rule (b) is similar except the condition is that $p(\overline{X}, \overline{a})$ needs to be true in *some answer set* of $M(\bar{b})$. Rule (c) numbers the answer sets of $M(\bar{b})$ and tuple $(i, \overline{X})$ is imported if $p(\overline{X}, \overline{a})$ is true in the $i$th answer set of $M(\bar{b})$. An *ASP module* is defined as a quadruple of a module name, a set of parameters, a collection of normal rules and a collection of import rules. Semantics is only defined for modular programs with acyclic dependency graph, and answer sets of a module are defined with respect to the modular ASP program containing it. Also, it is required that import rules importing from the same module always have the same form. There is a prototype implementation[3] of the module system [68] and it has been used to solve a scheduling problem.

Programming-in-the-large approaches to modularity in ASP are mostly based on Lifschitz and Turner's splitting set theorem [39] or are variants of it. The splitting set theorem is covered in detail in Section 2.2.1, and we only discuss a module system based on splitting sets on a general level. The class of logic programs considered in [39] is that of *extended disjunctive logic programs*, that is, disjunctive logic programs with two kinds of negation. A *component structure* induced by a *splitting sequence*, that is, iterated splittings of a program, allows a bottom-up computation of answer sets. The restriction induced is that the dependency graph of the component chain needs to be acyclic.

[3]See `http://www.public.asu.edu/~tng01/modules_asp.html`.

Eiter, Gottlob, and Mannila [10] consider *disjunctive logic programs as a query language for relational databases*. A query program $\pi$ is instantiated with respect to an input database $D$ confined by an input schema $\mathbf{R}$. The semantics of $\pi$ determines, for example, the answer sets of $\pi[D]$ which are projected with respect to an output schema $\mathbf{S}$. Module architecture is based on both *positive and negative dependencies* and no recursion between modules is tolerated. These constraints enable a straightforward generalization of the splitting set theorem for the architecture.

Faber et al. [15] apply the *magic set method* in the evaluation of *Datalog programs with negation*, that is, effectively normal logic programs. This involves the concept of an *independent set* $S$ of a program $P$ which is a specialization of a splitting set. Due to close relationship to splitting sets, the flexibility of independent sets for parceling programs is limited in the same way.

## 1.2 OUR DESIGN CRITERIA AND GOALS

Following the ideas of *compositionality* and *full abstraction* originating from O'Keefe's ideas [59] we adopt the following design criteria for modularity within answer set programming.

- Communication between modules is managed through an *input/output interface*.
- *Module composition* operator $\oplus$ is suitably restricted to ensure that answer sets for individual modules can be combined into an answer set for the composition of modules.
- To go beyond the splitting set theorem [39] (some) *recursion* between modules is tolerated.
- *Equivalence relation* for modules ($\equiv_{\mathrm{m}}$) is defined in such a way that it reduces to *weak equivalence for programs with completely specified input*.
- Relation $\equiv_{\mathrm{m}}$ is a *congruence for* $\oplus$, that is, $P \equiv_{\mathrm{m}} Q$ implies

  $$P \oplus R \equiv_{\mathrm{m}} Q \oplus R$$

  for all modules $R$ for which $P \oplus R$ and $Q \oplus R$ are defined.

Our design superficially resembles that of Gaifman and Shapiro [17] but to guarantee compositionality and full abstraction properties for ASP programs, that is, for *logic programs under the stable model semantics* [20] special module conditions for module composition need to be incorporated. Note, that in answer set programming the semantics is based on answer sets, or more specifically on so-called *stable models* whereas in conventional logic programming, queries and logical consequences are of interest.

## 1.3 CONTRIBUTIONS

- We propose a simple and intuitive *notion for a logic program module* that interacts through an input/output interface. We define so-called *normal logic program modules* first and then extend the concepts to a more general class of SMODELS *program modules*.

- We show that the module system is fully compatible with the stable model semantics. This is achieved by restricting the *composition* of modules so that module-level stability implies program-level stability, and vice versa. In fact, our *module theorem* is a proper strengthening of the splitting set theorem [39] for the classes of logic programs considered in this report as our result allows negative recursion between modules.

- We introduce a notion of *modular equivalence* that is a proper *congruence relation for composition of modules* and analyze the computational complexity of deciding modular equivalence.

- We extend the *translation-based method* for verifying visible equivalence proposed in [31] to cover *verification of modular equivalence* of SMODELS program modules.

- We propose a method for modularizing the verification of visible equivalence and consider questions involved in finding a suitable module structure for a program in case in which there is no explicit a priori knowledge on the underlying structure.

- We present an experimental evaluation of the efficiency of the translation-based method in the verification of modular equivalence.

The results for normal logic program modules presented in Chapters 3 and 4 are published in [57, 58].

## 1.4 STRUCTURE OF THE REPORT

The rest of this report is organized as follows. In Chapter 2 the stable model semantics of normal logic programs and a variety of equivalence relations are presented as preliminaries for further elaboration. In Chapter 3 a notion of a logic program module is established and it is shown that full compatibility with the stable model semantics is achieved. An equivalence relation for modules called *modular equivalence* is introduced in Chapter 4. The results from Chapters 3 and 4 are extended to cover a more general class of SMODELS programs [65] in Chapter 5. In Chapter 6 the translation-based method for verifying equivalence proposed in [31] is extended to cover the verification of modular equivalence. Furthermore, a strategy for modularizing the verification of equivalence is proposed. Experimental evaluation of the efficiency of the translation-based method for verification of modular equivalence is presented in Chapter 7. The work is concluded in Chapter 8 with a discussion of further work.

# 2 PRELIMINARIES

We start this chapter by introducing the *stable model semantics* for *normal logic programs* in Section 2.1. Further properties of logic programs are discussed in Section 2.2. In Section 2.3 we review a variety of equivalence relations suggested for logic programs.

## 2.1 STABLE MODEL SEMANTICS

In this section we briefly go through the stable model semantics for *normal logic programs*.

**Definition 2.1** *A normal logic program (NLP) is a (finite) set of rules of the form*

$$h \leftarrow a_1, \ldots, a_n, {\sim}b_1, \ldots, {\sim}b_m, \tag{2.1}$$

*where $n \geq 0$, $m \geq 0$, and $h$, each $a_i$, and each $b_j$ are propositional atoms.*

Since the order of the atoms in a rule is not significant, we use a shorthand

$$h \leftarrow B^+, {\sim}B^-,$$

where $B^+ = \{a_1, \ldots, a_n\}$ and $B^- = \{b_1, \ldots, b_m\}$ and ${\sim}B = \{{\sim}b \mid b \in B\}$ for any set of propositional atoms $B$. The symbol "$\sim$" denotes *default negation* or *negation as failure to prove* which differs from classical negation [21]. Atoms $a$ and their default negations ${\sim}a$ are called *default literals*. A rule consists of two parts: $h$ is the *head* of the rule, and the rest is called the *body* of the rule. Furthermore set $B^+$ is called the *positive body* and set $B^-$ the *negative body*. We define $\mathrm{Body}^+(r) = B^+$ and $\mathrm{Body}^-(r) = B^-$ for a rule $r$ of the form (2.1). The *set of head atoms for a set of rules $P$* is defined as

$$\mathrm{Head}(P) = \{h \mid h \leftarrow B^+, {\sim}B^- \in P\}.$$

If the sets $P$ and $\mathrm{Head}(P)$ are singletons, we omit the braces for the sake of clarity, that is, for a rule $r = h \leftarrow B^+, {\sim}B^-$, we write $\mathrm{Head}(r) = h$ instead of $\mathrm{Head}(\{r\}) = \{h\}$. The semantics of rules is defined formally in Definition 2.3, but informally speaking, a head atom $h$ can be inferred if all the atoms in the positive body $B^+$ are inferable and all the atoms in the negative body $B^-$ are non-inferable. If the body of a rule is empty, the rule is called a *fact* and the symbol "$\leftarrow$" can be dropped. If $B^- = \emptyset$, the rule is *positive*. A program consisting only of positive rules is called a *positive logic program*.

Usually the *Herbrand base* $\mathrm{Hb}(P)$ of a normal logic program $P$ is defined to be the set of atoms appearing in the rules of $P$. We, however, use a revised definition: $\mathrm{Hb}(P)$ *is any finite fixed set of atoms containing all atoms appearing in the rules of $P$*. Under this definition the Herbrand base of a program $P$ can be extended by atoms having no occurrences in $P$. This aspect is useful, for example, when $P$ is obtained as a result of optimization and there is a need to keep track of the original Herbrand base. For instance, see Example 2.4 in the following.

We follow the ideas from [28], and partition $\mathrm{Hb}(P)$ into two parts $\mathrm{Hb}_{\mathrm{v}}(P)$ and $\mathrm{Hb}_{\mathrm{h}}(P)$ which determine the *visible* and the *hidden* parts of $\mathrm{Hb}(P)$, respectively. Visible atoms can be seen as an interface for interaction between programs, and hidden atoms are local to each program. Visibility aspects will be taken into account later when the notion of so-called *visible equivalence* (Definition 2.14) is introduced.

Given a logic program $P$, an *interpretation* $M$ is a subset of $\mathrm{Hb}(P)$ defining which of the atoms in $\mathrm{Hb}(P)$ are *true* ($a \in M$) and which are *false* ($a \notin M$).

**Definition 2.2** *Given a logic program $P$ and an interpretation $M \subseteq \mathrm{Hb}(P)$, $M$ is a (classical) model of $P$, denoted by $M \models P$ if and only if $B^+ \subseteq M$ and $B^- \cap M = \emptyset$ imply $h \in M$ for each rule $h \leftarrow B^+, {\sim}B^- \in P$.*

The semantics of positive programs is usually defined in terms of *least models* [45]. A model $M$ of $P$ is *minimal* if there is no interpretation $M' \models P$ such that $M' \subset M$. Every positive program $P$ has a unique minimal model [45], called the *least model* of $P$, and denoted by $\mathrm{LM}(P)$.

*Stable models* as proposed by Gelfond and Lifschitz [20] generalize least models for normal logic programs.

**Definition 2.3** *Given a normal logic program $P$ and a model candidate $M \subseteq \mathrm{Hb}(P)$ the Gelfond-Lifschitz reduct $P^M$ is*

$$P^M = \{h \leftarrow B^+ \mid h \leftarrow B^+, {\sim}B^- \in P \text{ and } M \cap B^- = \emptyset\}$$

*and $M$ is a stable model of $P$, if and only if $M = \mathrm{LM}(P^M)$.*

Stable models are not necessarily unique; a normal logic program may in general have several stable models or no stable models at all. The *set of stable models* of a normal logic program $P$ is denoted by $\mathrm{SM}(P)$.

**Example 2.4** Consider a normal logic program $P = \{a \leftarrow {\sim}b\}$ with $\mathrm{Hb}(P) = \mathrm{Hb}_{\mathrm{v}}(P) = \{a, b\}$. For $M_1 = \emptyset$ we get $P^{M_1} = \{a.\}$. Now, $M_1$ is not a stable model of $P$, since $M_1 \neq \{a\} = \mathrm{LM}(P^{M_1})$. For $M_2 = \{a\}$ we get $P^{M_2} = \{a.\}$. Now, $M_2 = \mathrm{LM}(P^{M_2})$, and $M_2 \in \mathrm{SM}(P)$. Furthermore it is easy to see that $M_2$ is the only stable model of $P$ and thus $\mathrm{SM}(P) = \{M_2\}$. Since there are no rules for $b$, one may consider a simplification $Q = \{a.\}$ for $P$. The interpretation $M_2$ is also the only stable model of $Q$. To keep track of atom $b$, we may then define $\mathrm{Hb}(Q) = \mathrm{Hb}_{\mathrm{v}}(Q) = \{a, b\}$. ■

## 2.2 PROPERTIES OF LOGIC PROGRAMS

In this section we go through some properties of logic programs that will be useful later on.

### 2.2.1 Splitting Sets

We formulate the *splitting set theorem* [39] for normal programs under the stable model semantics. The splitting set theorem can be used to simplify the computation of stable models by splitting a program into parts, and it is also a useful tool for structuring mathematical proofs for properties of logic programs.

**Definition 2.5** *A splitting set for a normal logic program $P$ is any set $U \subseteq \mathrm{Hb}(P)$ such that for every rule $h \leftarrow B^+, {\sim}B^- \in P$ it holds, that if $h \in U$ then $B^+ \cup B^- \subseteq U$.*

The set of rules $h \leftarrow B^+, {\sim}B^- \in P$ such that $\{h\} \cup B^+ \cup B^- \subseteq U$ is the *bottom* of $P$ relative to $U$, denoted by $\mathrm{b}_U(P)$. The set $\mathrm{t}_U(P) = P \setminus \mathrm{b}_U(P)$ is the *top* of $P$ relative to $U$ which can be partially evaluated with respect to an interpretation $X \subseteq U$. The result is a program $\mathrm{e}(\mathrm{t}_U(P), X)$ defined as

$$\{h \leftarrow (B^+ \setminus U), {\sim}(B^- \setminus U) \mid h \leftarrow B^+, {\sim}B^- \in \mathrm{t}_U(P), \\ B^+ \cap U \subseteq X \text{ and } (B^- \cap U) \cap X = \emptyset\}.$$

A *solution* to a program with respect to a splitting set is a pair consisting of a stable model $X$ for the bottom and a stable model $Y$ for the top partially evaluated with respect to $X$.

**Definition 2.6** *Given a splitting set $U$ for a normal logic program $P$, a solution to $P$ with respect to $U$ is a pair $\langle X, Y\rangle$ such that*

*(i) $X \subseteq U$ is a stable model of $\mathrm{b}_U(P)$, and*

*(ii) $Y \subseteq \mathrm{Hb}(P) \setminus U$ is a stable model of $\mathrm{e}(\mathrm{t}_U(P), X)$.*

Solutions and stable models relate as follows.

**Theorem 2.7** *(The splitting set theorem [39]). Let $U$ be a splitting set for a normal logic program $P$ and consider an interpretation $M \subseteq \mathrm{Hb}(P)$. Then $M \in \mathrm{SM}(P)$ if and only if the pair $\langle M \cap U, M \setminus U\rangle$ is a solution to $P$ with respect to $U$.*

The splitting set theorem can also be used in an iterative manner, if there is a *monotone sequence* of splitting sets $\{U_1, \ldots, U_i, \ldots\}$, that is, $U_i \subset U_j$ if $i < j$, for program $P$. This is called a *splitting sequence* and it induces a *component structure* for $P$. The splitting set theorem generalizes to a *splitting sequence theorem* [39], and given a splitting sequence, stable models of program $P$ can be computed iteratively bottom-up.

### 2.2.2 Dependency Relations

Given a normal logic program $P$ and $a, b \in \mathrm{Hb}(P)$, we say that $b$ *depends directly* on $a$, denoted $a \leq_1 b$, if and only if there is a rule $b \leftarrow B^+, {\sim}B^- \in P$ such that $a \in B^+$. The *positive dependency relation* $\leq \subseteq \mathrm{Hb}(P) \times \mathrm{Hb}(P)$ of $P$ is then defined as the *reflexive and transitive closure* of relation $\leq_1$. The *positive dependency graph* of $P$, denoted by $\mathrm{Dep}^+(P)$, is a graph with $\mathrm{Hb}(P)$ and $\{\langle b, a\rangle \mid a \leq_1 b\}$ as the sets of vertices and edges, respectively.

A *strongly connected component* (SCC) of $\mathrm{Dep}^+(P)$ is a maximal subset $C \subseteq \mathrm{Hb}(P)$ such that $a \leq b$ holds for all $a, b \in C$. The strongly connected components of $\mathrm{Dep}^+(P)$ partition $\mathrm{Hb}(P)$ into equivalence classes, that is, for an arbitrary strongly connected component of $\mathrm{Dep}^+(P)$, atoms $a, b \in C$ if and only if $a \leq b$ and $b \leq a$.

The dependency relation $\leq$ generalizes for strongly connected components: $C_i \leq C_j$, that is, $C_j$ depends positively on $C_i$, if and only if $c_i \leq c_j$ for any $c_i \in C_i$ and $c_j \in C_j$. We define a *strict partial order* $<_{\mathrm{p}}$ for the strongly connected components based on $\leq$: $C_i <_{\mathrm{p}} C_j$, if $C_i \leq C_j$ and $C_j \not\leq C_i$. We extend $<_{\mathrm{p}}$ into a *strict total order* by defining relation $<$ as follows:

- $C_i < C_j$ if $C_i <_{\mathrm{p}} C_j$;
- if $C_i \not<_{\mathrm{p}} C_j$ and $C_j \not<_{\mathrm{p}} C_i$, then we choose either $C_i < C_j$ or $C_j < C_i$ (but not both).

The strongly connected components of $\mathrm{Dep}^{+}(P)$ can also be used in order to partition normal logic program $P$. We say that a *rule defining an atom* $a \in \mathrm{Hb}(P)$ is a rule in $P$ in which $a$ appears as the head. Because of the minimality of stable models, if an atom $a$ belongs to a stable model $M$ of program $P$, then there has to be a rule $a \leftarrow B^{+}, \sim B^{-} \in P$ such that the body of the rule is satisfied by $M$. Thus, the rules in which the atom appears in the head are the ones that can give a justification for the atom belonging to a stable model. Let $\mathrm{Def}_P(a) = \{r \in P \mid \mathrm{Head}(r) = a\}$ denote the set of rules defining an atom $a \in \mathrm{Hb}(P)$. Furthermore, $P[C]$ denotes the *set of rules in $P$ defining a set of atoms $C \subseteq \mathrm{Hb}(P)$*, that is,

$$P[C] = \bigcup_{a \in C} \mathrm{Def}_P(a).$$

Now, given the strongly connected components $C_1, \ldots, C_n$ of $\mathrm{Dep}^{+}(P)$, the sets $P[C_i]$ partition $P$, that is,

$$P[C_i] \cap P[C_j] = \emptyset \text{ for all } i \neq j, \text{ and } \bigcup_{i=1}^{n} P[C_i] = P.$$

## 2.3 EQUIVALENCE RELATIONS FOR LOGIC PROGRAMS

A number of *equivalence relations* have been suggested for logic programs. We review some of these, restricting ourselves to the case of normal logic programs under the stable model semantics. Motivated by our design criteria for a module system, we are interested in congruence properties. We also consider the computational complexity involved in the problem of deciding equivalence of programs for the equivalence relations reviewed in this section.

### 2.3.1 Weak, Strong and Uniform Equivalence

Lifschitz, Pearce, and Valverde [37] address the notions of *weak/ordinary equivalence* and *strong equivalence*.

**Definition 2.8** *Normal logic programs $P$ and $Q$ are weakly equivalent, denoted by $P \equiv Q$, if and only if* $\mathrm{SM}(P) = \mathrm{SM}(Q)$*; and strongly equivalent, denoted by $P \equiv_{\mathrm{s}} Q$, if and only if $P \cup R \equiv Q \cup R$ for any normal logic program $R$.*

The program $R$ in the above definition can be understood as an arbitrary context in which the two programs being compared could be placed. Therefore strongly equivalent logic programs are semantics preserving substitutes of each other and relation $\equiv_{\mathrm{s}}$ is a *congruence relation* for $\cup$ among normal logic programs, that is, if $P \equiv_{\mathrm{s}} Q$, then $P \cup R \equiv_{\mathrm{s}} Q \cup R$ for all normal logic programs $R$. Using $R = \emptyset$ as context, one sees that $P \equiv_{\mathrm{s}} Q$ implies $P \equiv Q$. The converse does not hold in general, as the following example shows.

**Example 2.9** Consider programs $P = \{a \leftarrow {\sim}c.\}$ and $Q = \{a \leftarrow {\sim}b.\}$. Since $\mathrm{SM}(P) = \{\{a\}\} = \mathrm{SM}(Q)$, we have $P \equiv Q$. Choosing program $R = \{b \leftarrow {\sim}a.\}$ as the context to place $P$ and $Q$ in, shows that $P \not\equiv_{\mathrm{s}} Q$, as $\mathrm{SM}(P \cup R) = \{\{a\}\} \neq \{\{a\}, \{b\}\} = \mathrm{SM}(Q \cup R)$. ■

Example 2.9 also shows that weak equivalence fails to be a congruence relation for $\cup$, that is, $P \equiv Q$ does not imply $P \cup R \equiv Q \cup R$ in general.

Strong equivalence seems inappropriate for *fully modularizing* the verification task of weak equivalence because programs $P$ and $Q$ may be weakly equivalent even if they build on respective modules $P_i \subseteq P$ and $Q_i \subseteq Q$ that are not strongly equivalent. For the same reason, program transformations that are known to preserve strong equivalence [8] do not provide an inclusive basis for reasoning about weak equivalence. Nevertheless, there are cases where one can utilize the fact that strong equivalence implies weak equivalence. For instance, if $P$ and $Q$ are composed of strongly equivalent pairs of modules $P_i$ and $Q_i$ for all $i$, then $P$ and $Q$ can be directly inferred to be strongly and weakly equivalent.

A way to weaken the strong equivalence is to restrict possible contexts to sets of facts. The notion of *uniform equivalence* has its roots in the database community [64], see [7] for case of the stable model semantics.

**Definition 2.10** *Normal logic programs $P$ and $Q$ are uniformly equivalent, denoted by $P \equiv_{\mathrm{u}} Q$, if and only if $P \cup F \equiv Q \cup F$ for any set of facts $F$.*

Strong equivalence implies uniform equivalence, and uniform equivalence implies weak equivalence, but not vice versa (in both cases) as the following examples show. Example 2.11 also shows that uniform equivalence is not a congruence for $\cup$.

**Example 2.11** [8, Example 1] Consider normal logic programs $P = \{a.\}$ and $Q = \{a \leftarrow {\sim}b.\ a \leftarrow b.\}$. It holds $P \equiv_{\mathrm{u}} Q$, but $P \cup R \not\equiv Q \cup R$ for $R = \{b \leftarrow a.\}$. This implies $P \not\equiv_{\mathrm{s}} Q$ and $P \cup R \not\equiv_{\mathrm{u}} Q \cup R$. ■

**Example 2.12** Consider $P = \{a.\}$ and $Q = \{a \leftarrow {\sim}b.\}$. It holds $P \equiv Q$, since $\mathrm{SM}(P) = \{\{a\}\} = \mathrm{SM}(Q)$. They are not uniformly equivalent, that is, $P \not\equiv_{\mathrm{u}} Q$, since $\mathrm{SM}(P \cup \{b.\}) = \{\{a, b\}\} \neq \{\{b\}\} = \mathrm{SM}(Q \cup \{b.\})$. ■

The verification of weak equivalence forms a **coNP**-complete decision problem[4] for normal logic programs [49]. The same can be stated about the verification of uniform and strong equivalence (for $\equiv_{\mathrm{u}}$, see [7] and for $\equiv_{\mathrm{s}}$, [61, 40, 69]). For a survey on the computational complexity of equivalence verification for other classes of logic programs, see [9]. It is worth noticing that the computational complexity of deciding $\equiv$, $\equiv_{\mathrm{u}}$ and $\equiv_{\mathrm{s}}$ varies depending on the program class considered. For example, for the class of *disjunctive logic programs*, verifying $\equiv_{\mathrm{s}}$ is still a **coNP**-complete decision problem, but

[4]We assume that the reader is familiar with the basic concepts computational complexity, see, for example, [60] for an introduction.

verifying $\equiv$ as well as $\equiv_{\mathrm{u}}$ is on the second level of the polynomial hierarchy, that is, they are $\Pi_2^{\mathbf{P}}$-complete decision problems.

There are also *relativized variants of strong and uniform equivalence* [71], where the context is allowed to be constrained using a set of atoms $A$.

**Definition 2.13** *Normal logic programs $P$ and $Q$ are strongly equivalent relative to $A$, denoted by $P \equiv_{\mathrm{s}}^{A} Q$, if and only if $P \cup R \equiv Q \cup R$ for all normal logic programs $R$ over the set of atoms $A$; and uniformly equivalent relative to $A$, denoted by $P \equiv_{\mathrm{u}}^{A} Q$, if and only if $P \cup F \equiv Q \cup F$ for any set of facts $F \subseteq A$.*

Setting $A = \emptyset$ reduces both relativized notions to weak equivalence, and thus neither of them is a congruence for $\cup$ in general. Verification of $\equiv_{\mathrm{u}}^{A}/\equiv_{\mathrm{s}}^{A}$ is a **coNP**-complete decision problem for normal logic programs [71].

## 2.3.2 Visible Equivalence

For $P \equiv Q$ to hold, the stable models in $\mathrm{SM}(P)$ and $\mathrm{SM}(Q)$ have to be identical subsets of $\mathrm{Hb}(P)$ and $\mathrm{Hb}(Q)$, respectively. The same effect can be seen with $P \equiv_{\mathrm{s}} Q$ and $P \equiv_{\mathrm{u}} Q$. This makes these notions of equivalence less useful if $\mathrm{Hb}(P)$ and $\mathrm{Hb}(Q)$ differ by some (local) atoms which are not trivially false in all stable models. Such atoms may be needed when some auxiliary concepts are formalized using rules. For this reason, Janhunen introduces a slightly more general notion of equivalence [28] which tries to take the interfaces of logic programs properly into account. The key idea is that the Herbrand base of a program is divided into *visible* and *hidden parts*, and the hidden atoms in $\mathrm{Hb}_{\mathrm{h}}(P)$ and $\mathrm{Hb}_{\mathrm{h}}(Q)$ are considered to be local to $P$ and $Q$ and thus negligible as far as the equivalence of the programs is concerned.

**Definition 2.14** *Normal logic programs $P$ and $Q$ are visibly equivalent, denoted by $P \equiv_{\mathrm{v}} Q$, if and only if $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$ and there is a bijection $f : \mathrm{SM}(P) \rightarrow \mathrm{SM}(Q)$ such that for all interpretations $M \in \mathrm{SM}(P)$,*

$$M \cap \mathrm{Hb}_{\mathrm{v}}(P) = f(M) \cap \mathrm{Hb}_{\mathrm{v}}(Q).$$

Note that the number of stable models is preserved under $\equiv_{\mathrm{v}}$. Such a strict correspondence of models is much dictated by the answer set programming methodology: the stable models of a program usually correspond to the solutions of the problem being solved and thus the exact preservation of models is highly significant.

**Example 2.15** Consider normal logic programs

$$\begin{aligned} P &= \{a \leftarrow b.\ a \leftarrow c.\ b \leftarrow {\sim}c.\ c \leftarrow {\sim}b.\}, \text{ and} \\ Q &= \{d \leftarrow {\sim}b.\ b \leftarrow {\sim}d.\ a \leftarrow c.\ c.\} \end{aligned}$$

with $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q) = \{a, b\}$ and $\mathrm{Hb}_{\mathrm{h}}(P) = \mathrm{Hb}_{\mathrm{h}}(Q) = \{c, d\}$. The stable models of $P$ are $M_1 = \{a, b\}$ and $M_2 = \{a, c\}$ whereas for $Q$ they are $N_1 = \{a, b, c\}$ and $N_2 = \{a, c, d\}$. Thus $P \not\equiv Q$ is clearly the case, but we have a bijection $f : \mathrm{SM}(P) \rightarrow \mathrm{SM}(Q)$, which maps $M_i$ to $N_i$ for $i \in \{1, 2\}$, such that $M \cap \mathrm{Hb}_{\mathrm{v}}(P) = f(M) \cap \mathrm{Hb}_{\mathrm{v}}(Q)$. Thus $P \equiv_{\mathrm{v}} Q$ holds. ■

In the fully visible case, that is, for $\mathrm{Hb_h}(P) = \mathrm{Hb_h}(Q) = \emptyset$, the relation $\equiv_\mathrm{v}$ becomes very close to $\equiv$. The only difference is the requirement $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ insisted by $\equiv_\mathrm{v}$. This is of little importance as Herbrand bases can always be extended to meet $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$. Since weak equivalence is not a congruence for $\cup$, visible equivalence cannot be either a congruence for $\cup$.

The computational complexity of deciding $\equiv_\mathrm{v}$ is analyzed in [31]. If the use of hidden atoms is not limited in any way, the problem of verifying visible equivalence becomes at least as hard as the counting problem #SAT which is #**P**-complete [70]. It is possible, however, to govern the computational complexity by limiting the use of hidden atoms by the property of having *enough visible atoms* (the EVA property for short) [31]. Consider a normal logic program $P$ and a set of atoms $A \subseteq \mathrm{Hb}(P)$. Let $A_\mathrm{v} = A \cap \mathrm{Hb_v}(P)$ and $A_\mathrm{h} = A \cap \mathrm{Hb_h}(P)$ denote the *visible* and the *hidden parts* of $A$, respectively. In Definition 2.16 the hidden part of a logic program $P$ is extracted by partially evaluating $P$ with respect to an interpretation for its visible part.

**Definition 2.16** *Consider a normal logic program $P$ and an interpretation $M_\mathrm{v} \subseteq \mathrm{Hb_v}(P)$ for the visible part of $P$. The hidden part of $P$ relative to $M_\mathrm{v}$, denoted by $P_\mathrm{h}/M_\mathrm{v}$, is*

$$\begin{aligned} P_\mathrm{h}/M_\mathrm{v} = \{h \leftarrow B_\mathrm{h}^+, {\sim}B_\mathrm{h}^- \mid\ & h \leftarrow B^+, {\sim}B^- \in P, h \in \mathrm{Hb_h}(P), \\ & B_\mathrm{v}^+ \subseteq M_\mathrm{v} \text{ and } B_\mathrm{v}^- \cap M_\mathrm{v} = \emptyset\}. \end{aligned}$$

This construction can be seen as the simplification operation $\mathrm{simp}(P, T, F)$ proposed by Cholewinski and Truszczyński [6], restricted in the sense that $T$ and $F$ are subsets of $\mathrm{Hb_v}(P)$ rather than $\mathrm{Hb}(P)$. More precisely,

$$P_\mathrm{h}/M_\mathrm{v} = \mathrm{simp}(P, M_\mathrm{v}, \mathrm{Hb_v}(P) - M_\mathrm{v})$$

for a normal logic program $P$. Now, the property of having enough visible atoms, that is, the EVA property, is defined as follows.

**Definition 2.17** *A normal logic program $P$ has enough visible atoms if and only if $P_\mathrm{h}/M_\mathrm{v}$ has a unique stable model for every $M_\mathrm{v} \subseteq \mathrm{Hb_v}(P)$.*

Here the intuition is that the interpretation of $\mathrm{Hb_h}(P)$ is uniquely determined for each interpretation of $\mathrm{Hb_v}(P)$ if $P$ has the EVA property. Consequently, the stable models of $P$ can be distinguished on the basis of their visible parts.

**Example 2.18** [31, Example 4.15] Consider normal logic programs

$$\begin{aligned} P &= \{a \leftarrow b.\}, \\ Q &= \{a \leftarrow c.\ c \leftarrow b.\}, \text{ and} \\ R &= \{a \leftarrow {\sim}c.\ c \leftarrow {\sim}d.\ d \leftarrow b.\} \end{aligned}$$

with $\mathrm{Hb_v}(P) = \mathrm{Hb_v}(Q) = \mathrm{Hb_v}(R) = \{a, b\}$. Given $I_\mathrm{v} = \emptyset$, the hidden parts are $P_\mathrm{h}/I_\mathrm{v} = \emptyset$, $Q_\mathrm{h}/I_\mathrm{v} = \emptyset$, and $R_\mathrm{h}/I_\mathrm{v} = \{c \leftarrow {\sim}d.\}$ for which unique stable models $M_P = M_Q = \emptyset$ and $M_R = \{c\}$ are obtained. On the other hand, we obtain $P_\mathrm{h}/J_\mathrm{v} = \emptyset$, $Q_\mathrm{h}/J_\mathrm{v} = \{c.\}$, and $R_\mathrm{h}/J_\mathrm{v} = \{c \leftarrow {\sim}d.\ d.\}$ for $J_\mathrm{v} = \{a, b\}$. Thus the respective unique stable models of the hidden parts are $N_P = \emptyset$ and $N_Q = \{c\}$, and $N_R = \{d\}$. ■

Although verifying the EVA property can be hard in general [31, Proposition 4.14], there are syntactic subclasses of normal programs (for example those for which $P_\mathrm{h}/M_\mathrm{v}$ is always *stratified*) with the EVA property. The use of visible atoms remains unlimited and thus the full expressiveness of normal logic programs remains basically available. Also note that the EVA property can always be achieved by declaring sufficiently many atoms visible. For SMODELS logic programs[5] with the EVA property, the verification of visible equivalence is a **coNP**-complete decision problem [31].

### 2.3.3 General Equivalence Frameworks

Eiter, Tompits, and Woltran [12] have introduced a very general framework based on *equivalence frames* to capture various kinds of equivalence relations. All the equivalence relations defined in Section 2.3.1 can be defined using the framework. Visible equivalence, however, is exceptional in the sense that it does not fit into equivalence frames based on *projected answer sets*. A projective variant of Definition 2.14 defines a weaker notion of equivalence called *weak visible equivalence* [28].

**Definition 2.19** *Normal logic programs $P$ and $Q$ are weakly visibly equivalent, denoted by $P \equiv_\mathrm{w} Q$, if and only if* $\mathrm{Hb}_\mathrm{v}(P) = \mathrm{Hb}_\mathrm{v}(Q)$ *and*

$$\{M \cap \mathrm{Hb}_\mathrm{v}(P) \mid M \in \mathrm{SM}(P)\} = \{N \cap \mathrm{Hb}_\mathrm{v}(Q) \mid N \in \mathrm{SM}(Q)\}.$$

As a consequence, the number of answer sets may not be preserved which is somewhat unsatisfactory because of the general nature of answer set programming as discussed in the previous section.

**Example 2.20** [31, page 14] Consider programs $P = \{a \leftarrow {\sim}b.\ b \leftarrow {\sim}a.\ \}$ and $Q_n = P \cup \{c_i \leftarrow {\sim}d_i.\ d_i \leftarrow {\sim}c_i.\ \mid 0 < i \leq n\}$ with $\mathrm{Hb}_\mathrm{v}(P) = \mathrm{Hb}_\mathrm{v}(Q_n) = \{a, b\}$. Whenever $n > 0$ these programs are not visibly equivalent but they are weakly visibly equivalent. With sufficiently large values of $n$ it is no longer feasible to count the number of different stable models, that is, solutions, if $Q_n$ is used. ■

Note that $Q_n$ in Example 2.20 for $n > 0$ does not have the EVA property. However, under the EVA assumption weak visible equivalence coincides with visible equivalence.

[5]Normal logic programs are a subclass of SMODELS programs. We will introduce the class of SMODELS programs later in Chapter 5.

# 3 MODULAR LOGIC PROGRAMS

We introduce a *module system for normal logic programs* in Section 3.1. The conditions for the *composition of modules* introduced in Section 3.2 guarantee full compatibility with the stable model semantics as shown in Section 3.3. The main result in Section 3.3 is a *module theorem* showing that local stability implies global stability, and vice versa, as long as the stable models of the submodules are compatible. Finally, we compare our module system to previous approaches to modularity in Section 3.4.

## 3.1 SYNTAX OF MODULAR LOGIC PROGRAMS

We define a *logic program module* similarly to Gaifman and Shapiro [17], but consider the case of normal logic programs instead of positive (disjunctive) logic programs.[6]

**Definition 3.1** *A triple $\mathbb{P} = (P, I, O)$ is a (propositional logic program)* module*, if*

- *$P$ is a finite set of rules of the form (2.1);*
- *$I$ and $O$ are sets of propositional atoms and $I \cap O = \emptyset$; and*
- $\text{Head}(P) \cap I = \emptyset$.

The Herbrand base of module $\mathbb{P}$, denoted by $\text{Hb}(\mathbb{P})$, is the set of atoms appearing in $P$ combined with $I \cup O$. Intuitively the set $I$ defines the *input* of the module and the set $O$ is the *output*. The input and output atoms are considered to be *visible*, that is, the visible Herbrand base of module $\mathbb{P}$ is

$$\text{Hb}_\text{v}(\mathbb{P}) = I \cup O.$$

Notice that $I$ and $O$ can also contain atoms not appearing in the rules similarly to the possibility of having additional atoms in the Herbrand bases of normal logic programs. All other atoms are *hidden*, that is,

$$\text{Hb}_\text{h}(\mathbb{P}) = \text{Hb}(\mathbb{P}) \setminus \text{Hb}_\text{v}(\mathbb{P}).$$

The restrictions $I \cap O = \emptyset$ and $\text{Head}(P) \cap I = \emptyset$ in Definition 3.1 are quite intuitive. It is natural to assume the input and the output of a module to be distinct. To make sure that the semantics of input atoms is not redefined inside the module, it is required that input atoms do not appear in the heads of the rules in a module.

Intuitively, if a module has an empty input set $I = \emptyset$, its semantics is fully determined as there is no dependency on input. Thus a logic program module $\mathbb{P} = (P, \emptyset, O)$ is effectively a *normal logic program $P$*, with $\text{Hb}(P) = \text{Hb}(\mathbb{P})$ and $\text{Hb}_\text{v}(P) = O$.

[6]Module architecture introduced in this chapter is extended to the class of SMODELS programs in Chapter 5.

## 3.2 COMBINING MODULES

When designing a logic program consisting of several modules, it is necessary to define the conditions under which the modules can be combined together. We adopt an approach that resembles that of Gaifman and Shapiro [17]. However, to guarantee compatibility with the stable model semantics, we need to incorporate a further restriction denying positive recursion between modules.

Basically the *join operation* takes the union of the disjoint sets of rules in the modules involved, if the following restrictions are met. First, the output sets of the modules to be composed need to be disjoint. This way all the rules defining an atom, that is, the rules in which the particular atom appears as the head, are collected into one module. Second, the hidden part of each module needs to remain local. Third, any positive recursion needs to be inside the modules, that is, we forbid positive recursion between modules.

We start by defining formally what is meant by positive recursion between two modules.

**Definition 3.2** *Consider modules $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$, and let $C_1, \ldots, C_n$ be the strongly connected components of $\mathrm{Dep}^+(P_1 \cup P_2)$. There is a positive recursion between modules $\mathbb{P}_1$ and $\mathbb{P}_2$, if $C_i \cap O_1 \neq \emptyset$ and $C_i \cap O_2 \neq \emptyset$ for some $C_i$.*

The idea is that all inter-module dependencies go through the input/output interface of the modules, that is, the output of one module can serve as the input for another, and hidden atoms are local to each module. If there is a strongly connected component $C_i$ in $\mathrm{Dep}^+(P_1 \cup P_2)$ containing atoms from both $O_1$ and $O_2$, we know that, if programs $P_1$ and $P_2$ are combined, some output atom $a$ of $\mathbb{P}_1$ depends positively on some output atom $b$ of $\mathbb{P}_2$ which again depends positively on $a$. This yields a positive recursion.

The module composition operation *join* and the conditions under which it is defined are as follows.

**Definition 3.3** *Let $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$ be modules such that*

*(i)* $O_1 \cap O_2 = \emptyset$;

*(ii)* $\mathrm{Hb_h}(\mathbb{P}_1) \cap \mathrm{Hb}(\mathbb{P}_2) = \emptyset$ *and* $\mathrm{Hb_h}(\mathbb{P}_2) \cap \mathrm{Hb}(\mathbb{P}_1) = \emptyset$*; and*

*(iii) there is no positive recursion between $\mathbb{P}_1$ and $\mathbb{P}_2$.*

*If conditions in items (i)–(iii) hold, the join of $\mathbb{P}_1$ and $\mathbb{P}_2$, denoted by $\mathbb{P}_1 \sqcup \mathbb{P}_2$, is defined, and*

$$\mathbb{P}_1 \sqcup \mathbb{P}_2 = (P_1 \cup P_2, (I_1 \setminus O_2) \cup (I_2 \setminus O_1), O_1 \cup O_2).$$

Conditions (i) and (ii) in Definition 3.3 are the same as in the module system of Gaifman and Shapiro [17]. Note also that condition (i) is actually redundant as it is implied by condition (iii). In practice it is possible to circumvent condition (ii) using a suitable scheme, for example, based on module names, to rename the hidden atoms uniquely for each module. In the following we generally assume that each module has a uniquely named hidden Herbrand base.

It is straightforward to show that $\sqcup$ has the following properties:

- Identity: $\mathbb{P} \sqcup (\emptyset, \emptyset, \emptyset) = (\emptyset, \emptyset, \emptyset) \sqcup \mathbb{P} = \mathbb{P}$ for all modules $\mathbb{P}$.
- Commutativity: $\mathbb{P}_1 \sqcup \mathbb{P}_2 = \mathbb{P}_2 \sqcup \mathbb{P}_1$ for all modules $\mathbb{P}_1$ and $\mathbb{P}_2$ such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined.
- Associativity: $(\mathbb{P}_1 \sqcup \mathbb{P}_2) \sqcup \mathbb{P}_3 = \mathbb{P}_1 \sqcup (\mathbb{P}_2 \sqcup \mathbb{P}_3)$ for all modules $\mathbb{P}_1, \mathbb{P}_2$ and $\mathbb{P}_3$ such that all pairwise joins are defined.

Note that equality "=" used here denotes syntactical equality. The semantics of modules is discussed in the next section and semantical equivalence for modules is defined in Chapter 4. Also notice that $\mathbb{P} \sqcup \mathbb{P}$ is usually undefined, which is a difference to $\cup$ for which it holds $P \cup P = P$ for all programs $P$.

The following hold for $\mathbb{P}_1 \sqcup \mathbb{P}_2$. Since each atom is defined in one module, the sets of rules in $\mathbb{P}_1$ and $\mathbb{P}_2$ are distinct, that is, $P_1 \cap P_2 = \emptyset$. Also,

$$\begin{aligned} \mathrm{Hb}(\mathbb{P}_1 \sqcup \mathbb{P}_2) &= \mathrm{Hb}(\mathbb{P}_1) \cup \mathrm{Hb}(\mathbb{P}_2), \\ \mathrm{Hb}_\mathrm{v}(\mathbb{P}_1 \sqcup \mathbb{P}_2) &= \mathrm{Hb}_\mathrm{v}(\mathbb{P}_1) \cup \mathrm{Hb}_\mathrm{v}(\mathbb{P}_2), \text{ and} \\ \mathrm{Hb}_\mathrm{h}(\mathbb{P}_1 \sqcup \mathbb{P}_2) &= \mathrm{Hb}_\mathrm{h}(\mathbb{P}_1) \cup \mathrm{Hb}_\mathrm{h}(\mathbb{P}_2). \end{aligned}$$

For the intersections of Herbrand bases, the following hold under conditions (i) and (ii) in Definition 3.3:

$$\begin{aligned} \mathrm{Hb}_\mathrm{v}(\mathbb{P}_1) \cap \mathrm{Hb}_\mathrm{v}(\mathbb{P}_2) &= \mathrm{Hb}(\mathbb{P}_1) \cap \mathrm{Hb}(\mathbb{P}_2) \\ &= (I_1 \cap I_2) \cup (I_1 \cap O_2) \cup (I_2 \cap O_1) \qquad (3.1) \\ \mathrm{Hb}_\mathrm{h}(\mathbb{P}_1) \cap \mathrm{Hb}_\mathrm{h}(\mathbb{P}_2) &= \emptyset. \end{aligned}$$

Notice that the conditions in Definition 3.3 impose no restrictions on positive dependencies *inside* modules or on *negative* dependencies in general. The input of $\mathbb{P}_1 \sqcup \mathbb{P}_2$ can be smaller than the union of inputs of individual modules because $\mathbb{P}_1$ may provide input for $\mathbb{P}_2$, and conversely, as demonstrated below.

**Example 3.4** Consider modules

$$\begin{aligned} \mathbb{P} &= (\{a \leftarrow {\sim}b.\}, \{b\}, \{a\}), \text{ and} \\ \mathbb{Q} &= (\{b \leftarrow {\sim}a.\}, \{a\}, \{b\}). \end{aligned}$$

The join $\mathbb{P} \sqcup \mathbb{Q}$ is defined, and $\mathbb{P} \sqcup \mathbb{Q} = (\{a \leftarrow {\sim}b.\ b \leftarrow {\sim}a.\}, \emptyset, \{a, b\})$. ■

## 3.3 STABLE MODEL SEMANTICS FOR MODULES

The stable model semantics of a module is defined with respect to a given input, that is, a subset of the input atoms of the module. An input is seen as a set of facts (or a database) to be combined with the module.

**Definition 3.5** *Given a module $\mathbb{P} = (P, I, O)$ and a set of atoms $A \subseteq I$, the instantiation of $\mathbb{P}$ with an actual input $A$ is*

$$\mathbb{P}(A) = \mathbb{P} \sqcup \mathbb{F}_A,$$

*where $\mathbb{F}_A = (\{a. \mid a \in A\}, \emptyset, I)$.*

Note that $\mathbb{P}(A) = (P \cup \{a. \mid a \in A\}, \emptyset, I \cup O)$ is essentially a normal logic program with $I \cup O$ as the visible Herbrand base. Thus the stable model semantics generalizes naturally for modules. We identify $\mathbb{P}(A)$ with the respective set of rules $P \cup \mathrm{F}_A$, where $\mathrm{F}_A = \{a. \mid a \in A\}$. In the following $M \cap I$ acts as a particular input with respect to which the module is instantiated.

**Definition 3.6** *An interpretation $M \subseteq \mathrm{Hb}(\mathbb{P})$ is a (classical) model of module $\mathbb{P} = (P, I, O)$, denoted by $M \models \mathbb{P}$, if and only if $M \models P \cup \mathrm{F}_{M \cap I}$.*

**Definition 3.7** *An interpretation $M \subseteq \mathrm{Hb}(\mathbb{P})$ is a stable model of module $\mathbb{P} = (P, I, O)$, denoted by $M \in \mathrm{SM}(\mathbb{P})$, if and only if*

$$M = \mathrm{LM}(P^M \cup \mathrm{F}_{M \cap I}).$$

We define a concept of *compatibility* to describe when an interpretation $M_1 \subseteq \mathrm{Hb}(\mathbb{P}_1)$ can be combined with an interpretation $M_2 \subseteq \mathrm{Hb}(\mathbb{P}_2)$. This is exactly when $M_1$ and $M_2$ share the common (visible) part.

**Definition 3.8** *Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be modules, and consider $M_1 \subseteq \mathrm{Hb}(\mathbb{P}_1)$ and $M_2 \subseteq \mathrm{Hb}(\mathbb{P}_2)$. Now, $M_1$ and $M_2$ are compatible, if and only if*

$$M_1 \cap \mathrm{Hb_v}(\mathbb{P}_2) = M_2 \cap \mathrm{Hb_v}(\mathbb{P}_1).$$

If one considers $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$ such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined, the condition for compatibility can be reformulated by recalling (3.1):

$$\begin{aligned} &M_1 \cap (I_1 \cap I_2) = M_2 \cap (I_1 \cap I_2), \\ &M_1 \cap (I_1 \cap O_2) = M_2 \cap (I_1 \cap O_2), \text{ and} \\ &M_1 \cap (O_1 \cap I_2) = M_2 \cap (O_1 \cap I_2). \end{aligned}$$

Theorem 3.9 relates program-level stability with module-level stability, that is, if a program (module) consists of several submodules, its stable models are locally stable for the respective submodules; and on the other hand, local stability implies global stability as long as the stable models of the submodules are compatible.

**Theorem 3.9** *(Module theorem). Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be modules such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined. Now, $M \in \mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$ if and only if $M_1 = M \cap \mathrm{Hb}(\mathbb{P}_1) \in \mathrm{SM}(\mathbb{P}_1)$, $M_2 = M \cap \mathrm{Hb}(\mathbb{P}_2) \in \mathrm{SM}(\mathbb{P}_2)$, and $M_1$ and $M_2$ are compatible.*

Proof of Theorem 3.9 is given in Appendix A. It is worth noticing that condition (iii) in Definition 3.3 is not needed to show that global stability implies local stability. On the other hand, Example 3.10 shows that conditions (i) and (ii) in Definition 3.3 are not enough to guarantee that local stability implies global stability.

**Example 3.10** Consider modules $\mathbb{P}_1 = (\{a \leftarrow b.\}, \{b\}, \{a\})$ and $\mathbb{P}_2 = (\{b \leftarrow a.\}, \{a\}, \{b\})$ with $\mathrm{SM}(\mathbb{P}_1) = \mathrm{SM}(\mathbb{P}_2) = \{\emptyset, \{a, b\}\}$. The join of $\mathbb{P}_1$ and $\mathbb{P}_2$ is not defined because there is a positive recursion between $\mathbb{P}_1$ and $\mathbb{P}_2$. Notice, however, that conditions (i) and (ii) in Definition 3.3 are satisfied. If we consider the stable models of $\mathbb{P} = (\{a \leftarrow b.\ b \leftarrow a.\}, \emptyset, \{a, b\})$, we get $\mathrm{SM}(\mathbb{P}) = \{\emptyset\}$. Thus, even though $M_1 = \{a, b\} \in \mathrm{SM}(\mathbb{P}_1)$ and $M_2 = \{a, b\} \in \mathrm{SM}(\mathbb{P}_2)$ are compatible, the positive dependency between $a$ and $b$ excludes $\{a, b\}$ from $\mathrm{SM}(\mathbb{P})$. ■

Theorem 3.9 straightforwardly generalizes for modules consisting of several submodules. Consider a collection of modules $\mathbb{P}_1, \ldots, \mathbb{P}_n$ such that the join $\mathbb{P}_1 \sqcup \cdots \sqcup \mathbb{P}_n$ is defined (recall that $\sqcup$ is associative). We say that a collection of interpretations $\{M_1, \ldots, M_n\}$ for modules $\mathbb{P}_1, \ldots, \mathbb{P}_n$, respectively, is *compatible*, if and only if $M_i$ and $M_j$ are pairwise compatible for all $1 \leq i, j \leq n$.

**Corollary 3.11** *Let $\mathbb{P}_1, \ldots, \mathbb{P}_n$ be a collection of modules such that the join $\mathbb{P}_1 \sqcup \cdots \sqcup \mathbb{P}_n$ is defined. Now, $M \in \mathrm{SM}(\mathbb{P}_1 \sqcup \cdots \sqcup \mathbb{P}_n)$ if and only if $M_i = M \cap \mathrm{Hb}(\mathbb{P}_i) \in \mathrm{SM}(\mathbb{P}_i)$ for all $1 \leq i \leq n$, and the collection $\{M_1, \ldots, M_n\}$ is compatible.*

Although Corollary 3.11 enables the computation of stable models on a module-by-module basis, it leaves us the task of excluding mutually incompatible combinations of stable models.

**Example 3.12** Consider modules

$$\begin{aligned}
\mathbb{P}_1 &= (\{a \leftarrow {\sim}b.\}, \{b\}, \{a\}), \\
\mathbb{P}_2 &= (\{b \leftarrow {\sim}c.\}, \{c\}, \{b\}), \text{ and} \\
\mathbb{P}_3 &= (\{c \leftarrow {\sim}a.\}, \{a\}, \{c\}).
\end{aligned}$$

The join $\mathbb{P} = \mathbb{P}_1 \sqcup \mathbb{P}_2 \sqcup \mathbb{P}_3$ is defined, and

$$\mathbb{P} = (\{a \leftarrow {\sim}b.\ b \leftarrow {\sim}c.\ c \leftarrow {\sim}a.\}, \emptyset, \{a, b, c\}).$$

We have $\mathrm{SM}(\mathbb{P}_1) = \{\{a\}, \{b\}\}$, $\mathrm{SM}(\mathbb{P}_2) = \{\{b\}, \{c\}\}$, and $\mathrm{SM}(\mathbb{P}_3) = \{\{a\}, \{c\}\}$. To apply Corollary 3.11 for finding $\mathrm{SM}(\mathbb{P})$, one has to find a compatible triple of stable models $M_1$, $M_2$, and $M_3$ for $\mathbb{P}_1$, $\mathbb{P}_2$, and $\mathbb{P}_3$, respectively.

- Now $\{a\} \in \mathrm{SM}(\mathbb{P}_1)$ and $\{c\} \in \mathrm{SM}(\mathbb{P}_2)$ are compatible, since $\{a\} \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}_2) = \emptyset = \{c\} \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}_1)$. However, $\{a\} \in \mathrm{SM}(\mathbb{P}_3)$ is not compatible with $\{c\} \in \mathrm{SM}(\mathbb{P}_2)$, since $\{c\} \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}_3) = \{c\} \neq \emptyset = \{a\} \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}_2)$. On the other hand, $\{c\} \in \mathrm{SM}(\mathbb{P}_3)$ is not compatible with $\{a\} \in \mathrm{SM}(\mathbb{P}_1)$, since $\{a\} \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}_3) = \{a\} \neq \emptyset = \{c\} \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{P}_1)$.

- Also $\{b\} \in \mathrm{SM}(\mathbb{P}_1)$ and $\{b\} \in \mathrm{SM}(\mathbb{P}_2)$ are compatible, but $\{b\} \in \mathrm{SM}(\mathbb{P}_1)$ is incompatible with $\{a\} \in \mathrm{SM}(\mathbb{P}_3)$. Nor is $\{b\} \in \mathrm{SM}(\mathbb{P}_2)$ compatible with $\{c\} \in \mathrm{SM}(\mathbb{P}_3)$.

Therefore it is impossible to select $M_1 \in \mathrm{SM}(\mathbb{P}_1)$, $M_2 \in \mathrm{SM}(\mathbb{P}_2)$, and $M_3 \in \mathrm{SM}(\mathbb{P}_3)$ such that $\{M_1, M_2, M_3\}$ is compatible, which is understandable as $\mathrm{SM}(\mathbb{P}) = \emptyset$. ■

## 3.4 COMPARISON WITH EARLIER APPROACHES

Our module system resembles Gaifman and Shapiro's module system [17]. However, to make our system compatible with the stable model semantics we need to introduce a further restriction of denying positive recursion between modules. Also other propositions involve similar type of conditions for module composition. For example, Brogi et al. [4] employ visibility conditions that correspond to the condition (ii) in Definition 3.3. However, their approach covers only positive programs under the least model semantics. Maher [47] forbids all recursion between modules and considers Przymusinski's *perfect models* [63] rather than stable models. Etalle and Gabbrielli [14] restrict the composition of *constraint logic program* [27] modules with a condition that is close to ours: $\mathrm{Hb}(P) \cap \mathrm{Hb}(Q) \subseteq \mathrm{Hb}_{\mathrm{v}}(P) \cap \mathrm{Hb}_{\mathrm{v}}(Q)$ but no distinction between input and output is made, for example, $O_P \cap O_Q \neq \emptyset$ is allowed according to their definitions.

Approaches to modularity within answer set programming typically do not allow any recursion (negative or positive) between modules [10, 68, 39]. Theorem 3.9 (module theorem) is strictly stronger than the splitting set theorem [39] for normal logic programs. On one hand, a splitting of a program can be used as a basis for a module structure. If $U$ is a splitting set for a normal logic program $P$, then define

$$P = \mathbb{B} \sqcup \mathbb{T} = (\mathrm{b}_U(P), \emptyset, U) \sqcup (\mathrm{t}_U(P), U, \mathrm{Hb}(P) \setminus U).$$

It follows directly from Theorems 2.7 and 3.9 that $M_1 \in \mathrm{SM}(\mathbb{B})$ and $M_2 \in \mathrm{SM}(\mathbb{T})$ are compatible if and only if $\langle M_1, M_2 \setminus U\rangle$ is a solution for $P$ with respect to $U$. On the other hand, the splitting set theorem cannot be applied in a non-trivial way to $\mathbb{P} \sqcup \mathbb{Q}$ from Example 3.4, since neither $\{a\}$ nor $\{b\}$ is a splitting set for $\mathbb{P} \sqcup \mathbb{Q}$.

It should be noted that applying our module theorem by computing stable models for each submodule and finding then the compatible pairs, might not be preferable, especially in a situation in which the splitting set theorem is applicable. However, the module theorem can be used in a similar manner as the splitting set theorem. Then the bottom module acts as an input generator for the top module, and one can simply find the stable models for the top module instantiated with the stable models of the bottom module.

**Example 3.13** Consider a normal logic program

$$P = \{a \leftarrow {\sim}b.\ b \leftarrow {\sim}a.\ c \leftarrow a.\}.$$

The set $U = \{a, b\}$ is a splitting set for $P$, and therefore the splitting set theorem can be applied: $\mathrm{b}_U(P) = \{a \leftarrow {\sim}b.\ b \leftarrow {\sim}a.\}$ and $\mathrm{t}_U(P) = \{c \leftarrow a.\}$. Now $M_1 = \{a\}$ and $M_2 = \{b\}$ are the stable models of $\mathrm{b}_U(P)$, and we can evaluate the top with respect to $M_1$ and $M_2$, resulting in solutions $\langle M_1, \{c\}\rangle$ and $\langle M_2, \emptyset\rangle$, respectively.

On the other hand, $P$ can be seen as join of modules $\mathbb{P}_1 = (\mathrm{b}_U(P), \emptyset, U)$ and $\mathbb{P}_2 = (\mathrm{t}_U(P), U, \{c\})$. Note that $\mathbb{P}_1$ can be viewed as a composite module, that is,

$$\mathbb{P}_1 = (\{a \leftarrow {\sim}b.\}, \{b\}, \{a\}) \sqcup (\{b \leftarrow {\sim}a.\}, \{a\}, \{b\}).$$

This is a partitioning not allowed by the splitting set theorem. We have $\mathrm{SM}(\mathbb{P}_1) = \{M_1, M_2\}$ and $\mathrm{SM}(\mathbb{P}_2) = \{\emptyset, \{b\}, \{a, c\}, \{a, b, c\}\}$. Out of eight possible pairs only $\langle M_1, \{a, c\}\rangle$ and $\langle M_2, \{b\}\rangle$ are compatible.

It is possible to apply Theorem 3.9 similarly to the splitting set theorem. Once we have stable models for module $\mathbb{P}_1$, we can instantiate $\mathbb{P}_2$ with respect to the stable models of $\mathbb{P}_1$:

$$\mathbb{P}_2(M_1 \cap U) = (\mathrm{t}_U(P) \cup \mathrm{F}_{M_1 \cap U}, \emptyset, \{a, b, c\})$$

has a single stable model $\{a, c\}$, and $\{b\}$ is the unique stable model of $\mathbb{P}_2(M_2 \cap U) = (\mathrm{t}_U(P) \cup \mathrm{F}_{M_2 \cap U}, \emptyset, \{a, b, c\})$. ■

The latter strategy used in Example 3.13 works even if there is negative recursion between the modules.

**Example 3.14** Consider modules $\mathbb{P}_1$, $\mathbb{P}_2$, and $\mathbb{P}_3$ from Example 3.12. We want to find stable models of $\mathbb{P} = \mathbb{P}_1 \sqcup \mathbb{P}_2 \sqcup \mathbb{P}_3$ through instantiations of the submodules. Now, $\mathrm{SM}(\mathbb{P}_1) = \{\{a\}, \{b\}\}$, where $M_1 = \{a\}$ is obtained with empty input, and $M_2 = \{b\}$ is obtained with input $\{b\}$.

- We instantiate $\mathbb{P}_3$ with respect to input $M_1 \cap \{a\} = M_1$, and get $\mathrm{SM}(\mathbb{P}_3(M_1)) = \{\{a\}\}$. We continue by instantiating $\mathbb{P}_2$ with respect to $M_1 \cap \{c\} = \emptyset$, and get $\mathrm{SM}(\mathbb{P}_2(\emptyset)) = \{\{b\}\}$. Now, we notice that $\{b\}$ is not compatible with $M_1$, and thus it is not possible to find a compatible triple of stable models starting from $M_1$.
- Thus we have to choose $M_2$. By instantiating $\mathbb{P}_3$ with respect to $M_2 \cap \{a\} = \emptyset$ we get $\mathrm{SM}(\mathbb{P}_3(\emptyset)) = \{\{c\}\}$. We continue by instantiating $\mathbb{P}_2$ with respect to $\{c\}$, and get $\mathrm{SM}(\mathbb{P}_2(\{c\})) = \{\{c\}\}$. Now, $\{c\}$ is not compatible with $M_2$, and thus it is not possible to find a compatible triple of stable models starting from $M_2$ either.

Therefore, there is no compatible collection $\{M_1, M_2, M_3\}$ such that $M_1 \in \mathrm{SM}(\mathbb{P}_1)$, $M_2 \in \mathrm{SM}(\mathbb{P}_2)$ and $M_3 \in \mathrm{SM}(\mathbb{P}_3)$, and $\mathrm{SM}(\mathbb{P}) = \emptyset$. ■

Our theorem also strengthens a module theorem given in [28, Theorem 6.22] to cover normal programs that involve positive body literals, too. The independent sets of Faber et al. [15] push negative recursion inside modules which is unnecessary in view of our results. The module theorem presented in [15] is also weaker than Theorem 3.9.

# 4 MODULAR CONGRUENCE

We introduce a notion of *modular equivalence* that is a proper congruence relation for composition of modules and relate the notion of modular equivalence with previously suggested equivalence relations in Section 4.1. We address aspects of computational complexity of verifying modular equivalence in Section 4.2.

## 4.1 MODULAR EQUIVALENCE

The definition of *modular equivalence* combines features from relativized uniform equivalence [71] and visible equivalence [28].

**Definition 4.1** *Modules* $\mathbb{P} = (P, I_P, O_P)$ *and* $\mathbb{Q} = (Q, I_Q, O_Q)$ *are modularly equivalent, denoted by* $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$*, if and only if*

*(i)* $I_P = I_Q = I$ *and* $O_P = O_Q = O$*, and*

*(ii)* $\mathbb{P}(A) \equiv_{\mathrm{v}} \mathbb{Q}(A)$ *for all* $A \subseteq I$.

Modular equivalence is very close to visible equivalence defined for modules. As a matter a fact, if Definition 2.14 is generalized for logic program modules, condition (ii) in Definition 4.1 can be revised to $\mathbb{P} \equiv_{\mathrm{v}} \mathbb{Q}$. However, $\mathbb{P} \equiv_{\mathrm{v}} \mathbb{Q}$ is not enough to cover condition (i) in Definition 4.1, as visible equivalence only enforces $\mathrm{Hb}_{\mathrm{v}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q})$. In a restricted case $I = \emptyset$ modular equivalence coincides with visible equivalence. To the other extreme, if $O = \emptyset$, equivalence $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ means that $\mathbb{P}$ and $\mathbb{Q}$ have the same number of stable models on each input.

Furthermore, if one considers the *fully visible case*, that is, the restriction $\mathrm{Hb}_{\mathrm{h}}(\mathbb{P}) = \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q}) = \emptyset$, modular equivalence can be seen as a special case of $A$-uniform equivalence for $A = I$. Recall, however, the restrictions $\mathrm{Head}(P) \cap I = \mathrm{Head}(Q) \cap I = \emptyset$ imposed by module structure. With a further restriction $I = \emptyset$, modular equivalence coincides with weak equivalence because $\mathrm{Hb}(\mathbb{P}) = \mathrm{Hb}(\mathbb{Q})$ can always be satisfied by extending Herbrand bases. Basically, setting $I = \mathrm{Hb}(\mathbb{P})$ would give us uniform equivalence, but the additional condition $\mathrm{Head}(P) \cap I = \emptyset$ leaves room for the empty module only.

Since $\equiv_{\mathrm{v}}$ is not a congruence relation for $\cup$, neither is modular equivalence. The situation changes, however, if one considers the join operator $\sqcup$ which suitably restricts possible contexts. Consider, for instance, programs $P = \{a.\}$ and $Q = \{a \leftarrow {\sim}b.\ a \leftarrow b.\}$ from Example 2.11. We can define modules based on them: $\mathbb{P} = (P, \{b\}, \{a\})$ and $\mathbb{Q} = (Q, \{b\}, \{a\})$. Now it is not possible to define a module $\mathbb{R}$ based on $R = \{b \leftarrow a.\}$ in a way that $\mathbb{Q} \sqcup \mathbb{R}$ is defined, and $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$.

**Theorem 4.2** *(Congruence) Let* $\mathbb{P}, \mathbb{Q}$ *and* $\mathbb{R}$ *be logic program modules. If* $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ *and both* $\mathbb{P} \sqcup \mathbb{R}$ *and* $\mathbb{Q} \sqcup \mathbb{R}$ *are defined, then* $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R}$.

In short, proof of Theorem 4.2 employs Theorem 3.9 and the definition of visible equivalence. A detailed proof of Theorem 4.2 is given in Appendix A.

**Corollary 4.3** *Let $\mathbb{P}, \mathbb{Q}, \mathbb{R}$ and $\mathbb{S}$ be logic program modules. If $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$, $\mathbb{R} \equiv_{\mathrm{m}} \mathbb{S}$, and $\mathbb{P} \sqcup \mathbb{R}$, $\mathbb{Q} \sqcup \mathbb{R}$, and $\mathbb{Q} \sqcup \mathbb{S}$ are defined, then $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{S}$.*

**Proof of Corollary 4.3** $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{R} \sqcup \mathbb{Q} \equiv_{\mathrm{m}} \mathbb{S} \sqcup \mathbb{Q} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{S}$, since syntactical equivalence $\mathbb{Q} \sqcup \mathbb{R} = \mathbb{R} \sqcup \mathbb{Q}$ (respectively $\mathbb{S} \sqcup \mathbb{Q} = \mathbb{Q} \sqcup \mathbb{S}$) implies modular equivalence. $\square$

It is instructive to consider a strong variant of modular equivalence defined in analogy to strong equivalence [37].

**Definition 4.4** *Modules $\mathbb{P}$ and $\mathbb{Q}$ are modularly strongly equivalent, denoted by $\mathbb{P} \equiv^{\mathrm{s}}_{\mathrm{m}} \mathbb{Q}$, if and only if $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R}$, for all $\mathbb{R}$ such that $\mathbb{P} \sqcup \mathbb{R}$ and $\mathbb{Q} \sqcup \mathbb{R}$ are defined.*

However, Theorem 4.2 implies that $\equiv^{\mathrm{s}}_{\mathrm{m}}$ removes nothing from $\equiv_{\mathrm{m}}$ since $\mathbb{P} \equiv^{\mathrm{s}}_{\mathrm{m}} \mathbb{Q}$ if and only if $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$.

## 4.2 ON COMPUTATIONAL COMPLEXITY

In this section we will make some observations about the computational complexity of verifying modular equivalence of normal logic program modules. In general, deciding $\equiv_{\mathrm{m}}$ is **coNP**-hard, since deciding the weak equivalence $P \equiv Q$ reduces to deciding $(P, \emptyset, \mathrm{Hb}(P)) \equiv_{\mathrm{m}} (Q, \emptyset, \mathrm{Hb}(Q))$. In the fully visible case, that is, for $\mathrm{Hb_h}(\mathbb{P}) = \mathrm{Hb_h}(\mathbb{Q}) = \emptyset$, deciding $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ for modules $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$ can be reduced to deciding relativized uniform equivalence $P \equiv^{I}_{\mathrm{u}} Q$ [71] and thus deciding $\equiv_{\mathrm{m}}$ is **coNP**-complete in this restricted case.

In the other extreme, if $\mathrm{Hb_v}(\mathbb{P}) = \mathrm{Hb_v}(\mathbb{Q}) = \emptyset$, then $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ if and only if $\mathbb{P}$ and $\mathbb{Q}$ have the same number of stable models. This suggests a much higher computational complexity of verifying $\equiv_{\mathrm{m}}$ in general because classical models can be captured with stable models [54] and counting stable models cannot be easier than #SAT which is #**P**-complete [70].

A way to govern the computational complexity of verifying $\equiv_{\mathrm{m}}$ is to limit the use of hidden atoms as done in the case of $\equiv_{\mathrm{v}}$ in [31]. By the EVA assumption [31], the verification of $\equiv_{\mathrm{v}}$ becomes **coNP**-complete for SMODELS programs[7] involving hidden atoms. We say that module $\mathbb{P} = (P, I, O)$ has enough visible atoms, if and only if $P$ has enough visible atoms with respect to $\mathrm{Hb_v}(P) = I \cup O$. We show in Theorem 4.5 that the problem of verifying $\equiv_{\mathrm{m}}$ can be reduced to that of $\equiv_{\mathrm{v}}$ by introducing a special module $\mathbb{G}_I$ that acts as a context generator in analogy to [71]. In the following,

$$\{\overline{a} \mid a \in I\} \cap \mathrm{Hb}(\mathbb{P}) = \{\overline{a} \mid a \in I\} \cap \mathrm{Hb}(\mathbb{Q}) = \emptyset,$$

that is, atoms $\overline{a}$ are new atoms not appearing in $\mathbb{P}$ or $\mathbb{Q}$.

**Theorem 4.5** *Consider modules $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$. Now $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ if and only if $\mathbb{P} \sqcup \mathbb{G}_I \equiv_{\mathrm{v}} \mathbb{Q} \sqcup \mathbb{G}_I$ where*

$$\mathbb{G}_I = (\{a \leftarrow \sim\overline{a}.\ \overline{a} \leftarrow \sim a \mid a \in I\}, \emptyset, I)$$

*generates all possible inputs for $\mathbb{P}$ and $\mathbb{Q}$.*

[7]The class of SMODELS programs includes normal logic programs as a subset.

**Proof of Theorem 4.5** Module $\mathbb{G}_I$ has $2^{|I|}$ stable models of the form

$$A \cup \{\overline{a} \mid a \in I \setminus A\}$$

where $A \subseteq I$. Thus $\mathbb{P} \equiv_{\mathrm{v}} \mathbb{P} \sqcup \mathbb{G}_I$ and $\mathbb{Q} \equiv_{\mathrm{v}} \mathbb{Q} \sqcup \mathbb{G}_I$ follow by Definitions 2.3 and 2.14 and Theorem 3.9. It follows that $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ if and only if $\mathbb{P}(A) \equiv_{\mathrm{v}} \mathbb{Q}(A)$ for all $A \subseteq I$ if and only if $\mathbb{P} \sqcup \mathbb{G}_I \equiv_{\mathrm{v}} \mathbb{Q} \sqcup \mathbb{G}_I$. $\square$

Generator module $\mathbb{G}_I$ has the EVA property. Consider arbitrary interpretation $M_{\mathrm{v}} \subseteq I$ for the visible part of $\mathrm{Hb}(\mathbb{G}_I) = I \cup \overline{I}$. The hidden part of $G_I = \{a \leftarrow {\sim}\overline{a}.\ \overline{a} \leftarrow {\sim}a \mid a \in I\}$ relative to $M_{\mathrm{v}}$ is $(G_I)_{\mathrm{h}}/M_{\mathrm{v}} = \{\overline{a}. \mid a \notin M_{\mathrm{v}}\}$, and it has a unique stable model.

Based on these observations we can conclude that verifying the modular equivalence of modules with the EVA property is a **coNP**-complete decision problem. We define language **EQM** as follows: for any (normal logic program) modules $\mathbb{P}$ and $\mathbb{Q}$, $\langle \mathbb{P}, \mathbb{Q} \rangle \in$ **EQM** if and only if $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$. Similarly, for any normal logic programs $P$ and $Q$, we say $\langle P, Q \rangle \in$ **EQ** if and only if $P \equiv Q$; and $\langle P, Q \rangle \in$ **EQV** if and only if $P \equiv_{\mathrm{v}} Q$. Recall that **EQ** and **EQV** (the latter when restricted to programs having enough visible atoms) are **coNP**-complete decision problems [49, 31].

**Corollary 4.6** **EQM** *is a* **coNP**-*complete decision problem for modules having enough visible atoms.*

**Proof of Corollary 4.6** Since Theorem 4.5 shows that **EQM** can be reduced to **EQV** and $\mathbb{G}_I$ has the EVA property, **EQM** $\in$ **coNP** for modules having enough visible atoms. The **coNP**-hardness follows by reduction from weak equivalence. For any normal logic programs $P$ and $Q$, $\langle P, Q \rangle \in$ **EQ** if and only if $\langle (P, \emptyset, \mathrm{Hb}(P)), (Q, \emptyset, \mathrm{Hb}(Q)) \rangle \in$ **EQM**. $\square$

# 5 MODULARITY FOR SMODELS PROGRAMS

In this chapter we extend the results obtained for normal logic program modules in Chapters 3 and 4 to cover a module system based on the class of SMODELS *programs* [65]. We start by briefly introducing the syntax and semantics of SMODELS programs in Section 5.1 and continue by defining SMODELS *program modules* and their properties, ending with a brief recapitulation on computational complexity in Section 5.2.

Instead of proving from scratch that the module theorem and the congruence property of modular equivalence also hold for SMODELS program modules we *translate* SMODELS program modules into normal logic program modules. We then show in Theorem 5.15 that the translation is *homomorphic under modular equivalence*, and it does not limit the possible compositions of modules. Furthermore, Theorem 5.16 shows that the translation *preserves modular equivalence*. Now, based on Theorems 5.15 and 5.16 we can directly generalize the results for normal logic program modules to the case of SMODELS program modules.

## 5.1 INTRODUCTION TO SMODELS PROGRAMS

The forms of rules used in SMODELS programs [65] are listed in Definition 5.1. Besides *basic rules* (5.1) used in *normal logic programs*, there are also *constraint rules* (5.2), *choice rules* (5.3), *weight rules* (5.4), and *compute statements* (5.5).

**Definition 5.1** *An* SMODELS *program is a finite set of rules of the forms*

$$h \leftarrow B^+, {\sim}B^- \tag{5.1}$$

$$h \leftarrow c\,\{B^+, {\sim}B^-\} \tag{5.2}$$

$$\{H\} \leftarrow B^+, {\sim}B^- \tag{5.3}$$

$$h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \tag{5.4}$$

$$\mathsf{compute}\ \{B^+, {\sim}B^-\} \tag{5.5}$$

*where $h$ is a propositional atom, $B^+$, $B^-$, and $H$ are sets of propositional atoms, $H \neq \emptyset$; and $W_{B^+}$ and $W_{B^-}$ are sets of natural numbers; and $w$ and $c$ are natural numbers.*

We define $\mathrm{Body}^+(r) = B^+$ and $\mathrm{Body}^-(r) = B^-$ for rules $r$ of the forms (5.1) – (5.4). In a weight rule (5.4) each atom $a \in B^+$ (respectively each $b \in B^-$) is associated with a weight $w_a \in W_{B^+}$ (respectively $w_b \in W_{B^-}$). Note that a constraint rule (5.2) is equivalent to a basic rule (5.1) if $c = |B^+| + |B^-|$. Also, a weight rule (5.4) reduces to a constraint rule (5.2) when all weights are equal to 1 and $w = c$. The satisfaction relation $M \models r$ generalizes for the rule types (5.2)–(5.5) as follows.

**Definition 5.2** *Given an interpretation $M \subseteq \mathrm{Hb}(P)$ for an* SMODELS *program $P$,*

- *a constraint rule (5.2) is satisfied in $M$ if and only if $c \leq |B^+ \cap M| + |B^- \setminus M|$ implies $h \in M$.*

- *a choice rule (5.3) is always satisfied in $M$,*
- *a weight rule (5.4) is satisfied in $M$ if and only if*

$$\begin{aligned} w &\leq \sum_{a \in B^+ \cap M} w_a + \sum_{b \in B^- \setminus M} w_b \\ &= \mathrm{WS}_M(B^+ = W_{B^+}, {\sim}B^- = W_{B^-}) \end{aligned} \tag{5.6}$$

  *implies $h \in M$, and*

- *a compute statement (5.5) is satisfied in $M$ if and only if $B^+ \subseteq M$ and $M \cap B^- = \emptyset$.*

The generalization of Gelfond-Lifschitz reduct for SMODELS programs is defined as follows.

**Definition 5.3** *For an* SMODELS *program $P$ and an interpretation $M \subseteq \mathrm{Hb}(P)$, the reduct $P^M$ contains*

1. *a basic rule $h \leftarrow B^+$ if and only if there is a basic rule (5.1) in $P$ such that $M \cap B^- = \emptyset$* **or** *there is a choice rule (5.3) in $P$ such that $h \in H \cap M$, and $M \cap B^- = \emptyset$;*
2. *a constraint rule $h \leftarrow c' \ \{B^+\}$ if and only if there is a constraint rule (5.2) in $P$ and $c' = \max(0, c - |B^- \setminus M|)$; and*
3. *a weight rule $h \leftarrow w' \leq \{B^+ = W_{B^+}\}$ if and only if there is a weight rule (5.4) in $P$ and $w' = \max(0, w - \mathrm{WS}_M({\sim}B^- = W_{B^-}))$.*

An SMODELS program $P$ is *positive* if each rule in $P$ is of the forms (5.1), (5.2) and (5.4) restricted to the case $B^- = \emptyset$. Given the least model semantics for positive programs the stable model semantics [20] straightforwardly generalizes for SMODELS programs. Similarly to the case of normal programs the reduction from Definition 5.3 is used, but the effect of compute statements must also be taken into account. Let $\mathrm{CompS}(P)$ denote the union of literals appearing in the compute statements (5.5) of $P$.

**Definition 5.4** *An interpretation $M \subseteq \mathrm{Hb}(P)$ is a stable model of an* SMODELS *program $P$ if and only if $M = \mathrm{LM}(P^M)$ and $M \models \mathrm{CompS}(P)$.*

**Example 5.5** Consider an SMODELS program

$$\begin{aligned} P = \{ & \{a, b\} \leftarrow {\sim}d. \\ & d \leftarrow {\sim}a, b. \\ & c \leftarrow 2 \leq \{a = 2, b = 1, d = 1\}. \\ & \mathsf{compute}\ \{c\}. \ \} \end{aligned}$$

Since $M \models \mathrm{CompS}(P)$ has to hold for a stable model, we only need consider interpretations such that $c \in M$. The reduct of $P$ with respect to $M_1 = \{c\}$ is $P^{M_1} = \{d \leftarrow b.\ c \leftarrow 2 \leq \{a = 2, b = 1, d = 1\}.\}$. Now $\mathrm{LM}(P^{M_1}) = \emptyset \neq M_1$ and thus $M_1 \notin \mathrm{SM}(P)$. The reduct of $P$ with respect to $M_2 = \{a, b, c\}$ is $P^{M_2} = \{a.\ b.\ c \leftarrow 2 \leq \{a = 2, b = 1, d = 1\}.\}$. Since $\mathrm{LM}(P^{M_2}) = M_2$ and $M_2 \models \mathsf{compute}\ \{c\}$, we have $M_2 \in \mathrm{SM}(P)$. Program $P$ has two stable models in total, that is, $M_3 = \{a, c\} \in \mathrm{SM}(P)$ in addition to $M_2$. ■

Definition 5.6 generalizes Definition 2.16 for SMODELS programs.

**Definition 5.6** *For an* SMODELS *program $P$ and an interpretation $M_\mathrm{v} \subseteq \mathrm{Hb}_\mathrm{v}(P)$, the hidden part of $P$ relative to $M_\mathrm{v}$, denoted by $P_\mathrm{h}/M_\mathrm{v}$, contains*

1. *$h \leftarrow B^+_\mathrm{h}, \sim B^-_\mathrm{h}$ if and only if there is a basic rule (5.1) in $P$ such that $h \in \mathrm{Hb}_\mathrm{h}(P)$, $B^+_\mathrm{v} \subseteq M_\mathrm{v}$ and $B^-_\mathrm{v} \cap M_\mathrm{v} = \emptyset$;*
2. *$\{H_\mathrm{h}\} \leftarrow B^+_\mathrm{h}, \sim B^-_\mathrm{h}$ if and only if there is a choice rule (5.3) in $P$ such that $H_h \neq \emptyset$, $B^+_\mathrm{v} \subseteq M_\mathrm{v}$ and $B^-_\mathrm{v} \cap M_\mathrm{v} = \emptyset$;*
3. *$h \leftarrow c' \ \{B^+_\mathrm{h}, \sim B^-_\mathrm{h}\}$ if and only if there is a constraint rule (5.2) in $P$ such that $h \in \mathrm{Hb}_\mathrm{h}(P)$ and $c' = \max(0, c - |B^+_\mathrm{v} \cap M_\mathrm{v}| - |B^-_\mathrm{v} \setminus M_\mathrm{v}|)$;*
4. *$h \leftarrow w' \leq \{B^+_\mathrm{h} = W_{B^+_\mathrm{h}}, \sim B^-_\mathrm{h} = W_{B^-_\mathrm{h}}\}$ if and only if there is a weight rule (5.4) in $P$ such that $h \in \mathrm{Hb}_\mathrm{h}(P)$ and*

$$w' = \max(0, w - \mathrm{WS}_{M_\mathrm{v}}(B^+_\mathrm{v} = W_{B^+_\mathrm{v}}, \sim B^-_\mathrm{v} = W_{B^-_\mathrm{v}})).$$

## 5.2 SMODELS PROGRAM MODULES

We define SMODELS program modules in analogy to modules based on normal logic programs. Let $\mathrm{Head}(P)$ denote the set of atoms appearing in the heads of rules (5.1) – (5.4) in an SMODELS program $P$.

**Definition 5.7** *A triple $\mathbb{P} = (P, I, O)$ is a (propositional)* SMODELS program module*, if*

- *$P$ is a finite set of rules of the forms (5.1) – (5.5);*
- *$I$ and $O$ are sets of propositional atoms and $I \cap O = \emptyset$; and*
- $\mathrm{Head}(P) \cap I = \emptyset$.

The Herbrand base of an SMODELS program module $\mathbb{P}$ is the set of atoms appearing in $P$ combined with $I \cup O$, and moreover, $\mathrm{Hb}_\mathrm{v}(\mathbb{P}) = I \cup O$ and $\mathrm{Hb}_\mathrm{h}(\mathbb{P}) = \mathrm{Hb}(\mathbb{P}) \setminus \mathrm{Hb}_\mathrm{v}(\mathbb{P})$.

In order to define the conditions for the composition of two SMODELS program modules we first define the *direct positive dependency relation* $\leq_1$ for SMODELS programs.

**Definition 5.8** *Consider an* SMODELS *program $P$. Given atoms $a, b \in \mathrm{Hb}(P)$, we say that $b$ depends directly on $a$, denoted by $a \leq_1 b$, if and only if there is*

- *a basic rule $b \leftarrow B^+, \sim B^- \in P$ such that $a \in B^+$, or*
- *a constraint rule $b \leftarrow c\,\{B^+, \sim B^-\} \in P$ such that $a \in B^+$ and $c \leq |B^+| + |B^-|$, or*
- *a choice rule $\{H\} \leftarrow B^+, \sim B^- \in P$ such that $b \in H$ and $a \in B^+$, or*
- *a weight rule $b \leftarrow w \leq \{B^+ = W_{B^+}, \sim B^- = W_{B^-}\} \in P$ such that $a \in B^+$ and $w \leq \sum_{c \in B^+} w_c + \sum_{d \in B^-} w_d$.*

Note the additional restrictions applied to constraint and weight rules in Definition 5.8. The restrictions can be justified by noticing that a head atom cannot depend on a positive body atom if the dependency is based on a rule the body of which can never be satisfied[8]. The *positive dependency relation* $\leq$ is defined as the reflective and transitive closure of $\leq_1$, and the *positive dependency graph* of an SMODELS program $P$ is a graph with $\mathrm{Hb}(P)$ and $\{\langle b, a\rangle \mid a \leq_1 b\}$ as the sets of vertices and edges, respectively.

The definition of positive recursion between modules, that is, Definition 3.2, can now be extended to SMODELS program modules. Also, the conditions under which two SMODELS program modules can be composed are exactly the same as given in Definition 3.3.

**Definition 5.9** *Let $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$ be* SMODELS *program modules such that*

*(i)* $O_1 \cap O_2 = \emptyset$;

*(ii)* $\mathrm{Hb}_\mathrm{h}(\mathbb{P}_1) \cap \mathrm{Hb}(\mathbb{P}_2) = \emptyset$ *and* $\mathrm{Hb}_\mathrm{h}(\mathbb{P}_2) \cap \mathrm{Hb}(\mathbb{P}_1) = \emptyset$*; and*

*(iii) there is no positive recursion between* $\mathbb{P}_1$ *and* $\mathbb{P}_2$.

*Then the join of* $\mathbb{P}_1$ *and* $\mathbb{P}_2$*, denoted as* $\mathbb{P}_1 \sqcup \mathbb{P}_2$*, is defined, and*

$$\mathbb{P}_1 \sqcup \mathbb{P}_2 = (P_1 \cup P_2, (I_1 \setminus O_2) \cup (I_2 \setminus O_1), O_1 \cup O_2).$$

The generalization of Definition 3.5, that is, an instantiation of an SMODELS program module with an actual input, gives us the means to define stable model semantics for SMODELS program modules in the natural way.

**Definition 5.10** *Given an* SMODELS *program module* $\mathbb{P} = (P, I, O)$ *and a set of atoms* $A \subseteq I$ *the instantiation of* $\mathbb{P}$ *with an actual input* $A$ *is*

$$\mathbb{P}(A) = \mathbb{P} \sqcup (\mathrm{F}_A, \emptyset, I).$$

**Definition 5.11** *An interpretation* $M \subseteq \mathrm{Hb}(\mathbb{P})$ *is a stable model of an* SMODELS *program module* $\mathbb{P} = (P, I, O)$*, denoted by* $M \in \mathrm{SM}(\mathbb{P})$*, if and only if* $M = \mathrm{LM}(P^M \cup \mathrm{F}_{M\cap I})$ *and* $M \models \mathrm{CompS}(P)$.

Instead of proving directly the generalization of Theorem 3.9 (the module theorem) for SMODELS program modules, we present a translation from SMODELS program modules to normal logic program modules, which then allows us to apply module theorem to SMODELS program modules, too. Before introducing the translation, we extend the notion of modular equivalence for SMODELS program modules.

**Definition 5.12** *Two* SMODELS *program modules* $\mathbb{P} = (P, I_P, O_P)$ *and* $\mathbb{Q} = (Q, I_Q, O_Q)$ *are modularly equivalent, denoted by* $\mathbb{P} \equiv_\mathrm{m} \mathbb{Q}$*, if and only if*

*(i)* $I_P = I_Q = I$ *and* $O_P = O_Q = O$*, and*

*(ii)* $\mathbb{P}(A) \equiv_\mathrm{v} \mathbb{Q}(A)$ *for all* $A \subseteq I$.

Since basic rules and constraint rules can be seen as special cases of weight rules, we only consider in the following SMODELS programs consisting of

[8]It could be possible to add further restrictions, for example on basic rules, but the definition given here suffices for our purposes.

choice rules (5.3), weight rules (5.4) and compute statements (5.5). Note that the translation presented in Definition 5.13 is in the worst case exponential with respect to the number of rules in the original module. For a more compact translation, see for example [16].

**Definition 5.13** *Given an* SMODELS *program module* $\mathbb{P} = (P, I, O)$*, its translation into a normal logic program module is defined as* $\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}) = (\mathrm{Tr}_{\mathrm{NLP}}(P), I, O)$*, where* $\mathrm{Tr}_{\mathrm{NLP}}(P)$ *contains the following rules:*

- *for each choice rule* $\{H\} \leftarrow B^+, {\sim}B^-$ *in* $P$ *the set of rules*

$$\{h \leftarrow B^+, {\sim}B^-, {\sim}\overline{h}.\ \ \overline{h} \leftarrow {\sim}h \mid h \in H\};$$

- *for each weight rule* $h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\}$ *in* $P$ *the set of rules*

$$\{h \leftarrow C, {\sim}D \mid C \subseteq B^+, D \subseteq B^- \text{ and } w \leq \sum_{c \in C} w_c + \sum_{d \in D} w_d\};\ \text{and}$$

- *for each compute statement* compute $\{B^+, {\sim}B^-\}$ *in* $P$ *the set of rules*

$$\{f_P \leftarrow {\sim}a, {\sim}f_P \mid a \in B^+\} \cup \{f_P \leftarrow b, {\sim}f_P \mid b \in B^-\};$$

*where each* $\overline{a}$ *and* $f_P$ *are new atoms not appearing in* $\mathrm{Hb}(\mathbb{P})$*, that is,*

$$\mathrm{Hb_h}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P})) = \mathrm{Hb_h}(\mathbb{P}) \cup \{f_P\} \cup \{\overline{a} \mid a \in H \text{ for } \{H\} \leftarrow B^+, {\sim}B^- \in P\}.$$

Notice that the new atoms introduced in $\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P})$ are hidden.

**Example 5.14** Consider an SMODELS program module

$$\mathbb{P} = (P, \emptyset, \{a, b, c, d\}),$$

where $P$ is the SMODELS program given in Example 5.5. Module $\mathbb{P}$ has two stable models, $M_1 = \{a, b, c\}$ and $M_2 = \{a, c\}$. Translation $\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}) = (\mathrm{Tr}_{\mathrm{NLP}}(P), \emptyset, \{a, b, c, d\})$ is a normal logic program module containing the following rules:

$$\left\{ \begin{array}{llll} a \leftarrow {\sim}d, {\sim}\overline{a}. & \overline{a} \leftarrow {\sim}a. & b \leftarrow {\sim}d, {\sim}\overline{b}. & \overline{b} \leftarrow {\sim}b. \\ d \leftarrow {\sim}a, b. & c \leftarrow a. & c \leftarrow a, b. & c \leftarrow a, d. \\ c \leftarrow b, d. & c \leftarrow a, b, d. & f_P \leftarrow {\sim}c, {\sim}f_p. & \end{array} \right\}.$$

The stable models of $\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P})$ are $N_1 = \{c, a, b\}$ and $N_2 = \{a, \overline{b}, c\}$, which correspond to $M_1$ and $M_2$ in Example 5.5, respectively. Thus we have shown $\mathbb{P} \equiv_{\mathrm{m}} \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P})$. ■

We show in Theorem 5.15 that the translation presented in Definition 5.13 does not limit the possible compositions of modules, and

$$\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2) \equiv_{\mathrm{m}} \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1 \sqcup \mathbb{P}_2),$$

that is, $\mathrm{Tr}_{\mathrm{NLP}}$ is *homomorphic* under modular equivalence.

**Theorem 5.15** *Consider* SMODELS *program modules $\mathbb{P}_1$ and $\mathbb{P}_2$. If $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined, then also* $\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2)$ *is defined and*

$$\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2) \equiv_{\mathrm{m}} \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1 \sqcup \mathbb{P}_2).$$

Proof of Theorem 5.15 is given in Appendix A. In Theorem 5.16 we show that the translation presented in Definition 5.13 is modularly equivalent.

**Theorem 5.16** $\mathbb{P} \equiv_{\mathrm{m}} \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P})$ *for any* SMODELS *program module $\mathbb{P}$.*

Proof of Theorem 5.16 is given in Appendix A. Notice that in Theorem 5.15 $\equiv_{\mathrm{m}}$ relates two normal logic program modules, that is, Definition 4.1 is used, and in Theorem 5.16 $\equiv_{\mathrm{m}}$ relates an SMODELS program module and a normal logic program module (a special case of an SMODELS program module), that is, Definition 5.12 is used.

Theorems 5.15 and 5.16 allow us to generalize the module theorem for SMODELS program modules.

**Theorem 5.17** *(Module theorem for* SMODELS *program modules) Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be* SMODELS *program modules such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined. Now, $M \in \mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$ if and only if $M_1 = M \cap \mathrm{Hb}(\mathbb{P}_1) \in \mathrm{SM}(\mathbb{P}_1)$, $M_2 = M \cap \mathrm{Hb}(\mathbb{P}_2) \in \mathrm{SM}(\mathbb{P}_2)$ and $M_1$ and $M_2$ are compatible.*

Proof of Theorem 5.17 is given in Appendix A. Similarly to the case of normal logic program modules, Theorem 5.17 directly generalizes to a collection of submodules.

**Corollary 5.18** *Let $\mathbb{P}_1, \ldots, \mathbb{P}_n$ be a collection of* SMODELS *program modules such that $\mathbb{P}_1 \sqcup \cdots \sqcup \mathbb{P}_n$ is defined. Then $M \in \mathrm{SM}(\mathbb{P}_1 \sqcup \cdots \sqcup \mathbb{P}_n)$ if and only if $M_i = M \cap \mathrm{Hb}(\mathbb{P}_i) \in \mathrm{SM}(\mathbb{P}_i)$ for all $1 \leq i \leq n$, and the collection $\{M_1, \ldots, M_n\}$ is compatible.*

If we apply Theorem 5.17 instead of Theorem 3.9 in the proof of Theorem 4.2, we see that the congruence property of modular equivalence generalizes for SMODELS program modules.

**Corollary 5.19** *Let $\mathbb{P}, \mathbb{Q}$ and $\mathbb{R}$ be* SMODELS *program modules. If $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ and both $\mathbb{P} \sqcup \mathbb{R}$ and $\mathbb{Q} \sqcup \mathbb{R}$ are defined, then $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R}$.*

Recall that deciding weak equivalence is a **coNP**-complete decision problem for the class of SMODELS programs, and deciding visible equivalence is a **coNP**-complete decision problem for SMODELS programs having enough visible atoms [31]. Thus it is straightforward to see that deciding modular equivalence for SMODELS program modules with the EVA property is a **coNP**-complete decision problem, too.

**Corollary 5.20** **EQM** *is a* **coNP***-complete decision problem for* SMODELS *program modules having enough visible atoms.*

# 6 VERIFYING MODULAR EQUIVALENCE

We extend the *translation-based method* used to verify visible equivalence in [31] to cover the *verification of modular equivalence* of SMODELS program modules in Section 6.1. We propose a method for modularizing the verification of visible equivalence in Section 6.2. In Section 6.3 we consider questions involved in the problem of finding a suitable module structure for a program for which there is no explicit a priori knowledge on the underlying structure.

## 6.1 TRANSLATION FOR VERIFYING MODULAR EQUIVALENCE

As a consequence of Theorem 4.5, the translation-based technique from [31, Corollary 5.9] can be used to verify $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ given that $\mathbb{P}$ and $\mathbb{Q}$ have enough visible atoms. Recall that $\mathbb{G}_I$ has the EVA property. More specifically, the task is to show that $\mathrm{EQT}(\mathbb{P} \sqcup \mathbb{G}_I, \mathbb{Q} \sqcup \mathbb{G}_I)$ and $\mathrm{EQT}(\mathbb{Q} \sqcup \mathbb{G}_I, \mathbb{P} \sqcup \mathbb{G}_I)$ have no stable models, where EQT is the translation function described in [31]. However, it seems that there is still room for improvement since the common context $\mathbb{G}_I$ is handled separately for the modules involved.

We start with a brief recapitulation of the structure of translation EQT proposed in [31]. Given two SMODELS programs $P$ and $Q$, the translation $\mathrm{EQT}(P, Q)$ is an SMODELS program consisting of four parts, that is,

$$\mathrm{EQT}(P, Q) = P \cup \mathrm{Hidden}^{\circ}(Q) \cup \mathrm{Least}^{\bullet}(Q) \cup \mathrm{UnStable}(Q).$$

The roles of individual parts of $\mathrm{EQT}(P, Q)$ in the search for a counter-example are the following.

- $P$ finds a stable model $M \in \mathrm{SM}(P)$.
- $\mathrm{Hidden}^{\circ}(Q)$ finds the unique stable model $N_{\mathrm{h}}$ for $Q_{\mathrm{h}}/M_{\mathrm{v}}$.
- $\mathrm{Least}^{\bullet}(Q)$ finds the least model of $Q^N$, where $N = M_{\mathrm{v}} \cup N_{\mathrm{h}}$.
- $\mathrm{UnStable}(Q)$ checks that $N \notin \mathrm{SM}(Q)$, that is, $N \neq \mathrm{LM}(Q^N)$ or $N \not\models \mathrm{CompS}(Q)$.

A more detailed discussion on the ideas behind the translation EQT is given in [31]. We present now a modified version of the translation adjusted to verification of modular equivalence.

**Definition 6.1** *Let* $\mathbb{P} = (P, I, O)$ *and* $\mathbb{Q} = (Q, I, O)$ *be* SMODELS *program modules having enough visible atoms. The translation*

$$\mathrm{EQT}(\mathbb{P}, \mathbb{Q}) = \mathbb{P} \sqcup \mathrm{Hidden}^{\circ}(\mathbb{Q}) \sqcup \mathrm{Least}^{\bullet}(\mathbb{Q}) \sqcup \mathrm{UnStable}(\mathbb{Q})$$

*combines* $\mathbb{P}$ *with three modules to be made precise by Definitions 6.2–6.4. Atoms c, d, and e introduced in Definition 6.4 are assumed to be new atoms not appearing in* $\mathrm{Hb}(\mathbb{P}) \cup \mathrm{Hb}(\mathbb{Q})$.

First thing to notice is that $\mathrm{EQT}(\mathbb{P},\mathbb{Q})$ is a module with input $I_{\mathrm{EQT}} = I$ and output $O_{\mathrm{EQT}} = O \cup O^{\bullet} \cup \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q})^{\bullet} \cup \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q})^{\circ} \cup \{c,d,e\}$. Here $A^{\bullet} = \{a^{\bullet} \mid a \in A\}$ and $A^{\circ} = \{a^{\circ} \mid a \in A\}$ for any set of atoms $A$; and each $a^{\bullet}$ and $a^{\circ}$ is a new atom not appearing in $\mathrm{Hb}(\mathbb{P}) \cup \mathrm{Hb}(\mathbb{Q})$. Furthermore $\mathrm{Hb}_{\mathrm{h}}(\mathrm{EQT}(\mathbb{P},\mathbb{Q})) = \mathrm{Hb}_{\mathrm{h}}(\mathbb{P})$.

The overall idea of the translation is the same as in [31]: rules of $\mathbb{P}$ capture a stable model $M$ for $\mathbb{P}$ while the remaining modules are used to ensure that $\mathbb{Q}$ does not have a stable model $N$ such that $M_{\mathrm{v}} = N_{\mathrm{v}}$. The translation $\mathrm{Hidden}^{\circ}(\cdot)$ to be introduced in Definition 6.2 contains exactly the same rules as [31, Definition 5.2]. We, however, define $\mathrm{Hidden}^{\circ}(\cdot)$ as a module to be able to take advantage of the module theorem in the correctness proof. The sets of rules in $\mathrm{Least}^{\bullet}(\cdot)$ and $\mathrm{UnStable}(\cdot)$ introduced in Definitions 6.3 and 6.4, respectively, are very similar to the sets of rules in [31, Definitions 5.3 and 5.4]. The difference is that the input/output interfaces of $\mathbb{P}$ and $\mathbb{Q}$ need to be taken into account, which results in smaller translations, and furthermore, $\mathrm{Least}^{\bullet}(\cdot)$ and $\mathrm{UnStable}(\cdot)$ are also defined as modules.

For modules with a completely specified input, that is, $\mathbb{P}$ and $\mathbb{Q}$ with $I = \emptyset$, the translation given in Definition 6.1 results in exactly the same set of rules as the one presented in [31].

**Definition 6.2** *The translation*

$$\mathrm{Hidden}^{\circ}(\mathbb{Q}) = (\mathrm{Hidden}^{\circ}(Q), I \cup O, \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q})^{\circ})$$

*of an* SMODELS *program module* $\mathbb{Q} = (Q, I, O)$ *contains*

1. *a basic rule* $h^{\circ} \leftarrow (B^{+}_{\mathrm{h}})^{\circ}, B^{+}_{\mathrm{v}}, {\sim}(B^{-}_{\mathrm{h}})^{\circ}, {\sim}B^{-}_{\mathrm{v}}$ *for each basic rule* $h \leftarrow B^{+}, {\sim}B^{-}$ *in* $Q$ *with* $h \in \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q})$*;*
2. *a constraint rule* $h^{\circ} \leftarrow c\ \{(B^{+}_{\mathrm{h}})^{\circ}, B^{+}_{\mathrm{v}}, {\sim}(B^{-}_{\mathrm{h}})^{\circ}, {\sim}B^{-}_{\mathrm{v}}\}$ *for each constraint rule* $h \leftarrow c\ \{B^{+}, {\sim}B^{-}\}$ *in* $Q$ *with* $h \in \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q})$*;*
3. *a choice rule* $\{H^{\circ}_{\mathrm{h}}\} \leftarrow (B^{+}_{\mathrm{h}})^{\circ}, B^{+}_{\mathrm{v}}, {\sim}(B^{-}_{\mathrm{h}})^{\circ}, {\sim}B^{-}_{\mathrm{v}}$ *for each choice rule* $\{H\} \leftarrow B^{+}, {\sim}B^{-}$ *in* $Q$ *with* $H_{\mathrm{h}} \neq \emptyset$*; and*
4. *a weight rule*

$$h^{\circ} \leftarrow w \leq \{(B^{+}_{\mathrm{h}})^{\circ} \cup B^{+}_{\mathrm{v}} = W_{B^{+}}, {\sim}((B^{-}_{\mathrm{h}})^{\circ} \cup B^{-}_{\mathrm{v}}) = W_{B^{-}}\}$$

   *for each weight rule* $h \leftarrow w \leq \{B^{+} = W_{B^{+}}, {\sim}B^{-} = W_{B^{-}}\}$ *in* $Q$ *with* $h \in \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q})$.

The translation $\mathrm{Hidden}^{\circ}(\mathbb{Q})$ includes rules that provide a representation for the hidden part $Q_{\mathrm{h}}/M_{\mathrm{v}}$ which depends dynamically on $M_{\mathrm{v}}$. This is achieved by leaving the visible atoms from $\mathrm{Hb}_{\mathrm{v}}(\mathbb{Q}) = \mathrm{Hb}_{\mathrm{v}}(\mathbb{P})$ untouched. The hidden parts of rules are renamed systematically using atoms from $\mathrm{Hb}_{\mathrm{h}}(\mathbb{Q})^{\circ}$. This is to capture the unique stable model $N_{\mathrm{h}}$ of $Q_{\mathrm{h}}/M_{\mathrm{v}}$ but renamed as $N^{\circ}_{\mathrm{h}}$.

For the following definitions we introduce shorthands $A_{\mathrm{o}} = A \cap O$ and $A_{\mathrm{i}} = A \cap I$ for any set of atoms $A \subseteq \mathrm{Hb}(\mathbb{P})$ and an SMODELS program module $\mathbb{P} = (P, I, O)$.

**Definition 6.3** *The translation*

$$\mathrm{Least}^{\bullet}(\mathbb{Q}) = (\mathrm{Least}^{\bullet}(Q), I \cup O \cup \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q})^{\circ}, O^{\bullet} \cup \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q})^{\bullet})$$

*of an* SMODELS *program module* $\mathbb{Q} = (Q, I, O)$ *contains*

1. *a rule* $h^\bullet \leftarrow B_\mathrm{i}^+, (B_\mathrm{o}^+)^\bullet, (B_\mathrm{h}^+)^\bullet, {\sim}B_\mathrm{v}^-, {\sim}(B_\mathrm{h}^-)^\circ$ *for each basic rule* $h \leftarrow B^+, {\sim}B^-$ *in* $Q$*;*

2. *a rule* $h^\bullet \leftarrow c\,\{B_\mathrm{i}^+, (B_\mathrm{o}^+)^\bullet, (B_\mathrm{h}^+)^\bullet, {\sim}B_\mathrm{v}^-, {\sim}(B_\mathrm{h}^-)^\circ\}$ *for each constraint rule* $h \leftarrow c\,\{B^+, {\sim}B^-\}$ *in* $Q$*;*

3. *a rule* $h^\bullet \leftarrow h, B_\mathrm{i}^+, (B_\mathrm{o}^+)^\bullet, (B_\mathrm{h}^+)^\bullet, {\sim}B_\mathrm{v}^-, {\sim}(B_\mathrm{h}^-)^\circ$ *for each choice rule* $\{H\} \leftarrow B^+, {\sim}B^-$ *in* $Q$ *and* $h \in H_\mathrm{v}$*;*
   *a rule* $h^\bullet \leftarrow h^\circ, B_\mathrm{i}^+, (B_\mathrm{o}^+)^\bullet, (B_\mathrm{h}^+)^\bullet, {\sim}B_\mathrm{v}^-, {\sim}(B_\mathrm{h}^-)^\circ$ *for each choice rule* $\{H\} \leftarrow B^+, {\sim}B^-$ *in* $Q$ *and* $h \in H_\mathrm{h}$*; and*

4. *a rule*

   $$h^\bullet \leftarrow w \leq \{B_\mathrm{i}^+ \cup (B_\mathrm{o}^+)^\bullet \cup (B_\mathrm{h}^+)^\bullet = W_{B^+}, {\sim}(B_\mathrm{v}^- \cup (B_\mathrm{h}^-)^\circ) = W_{B^-}\}$$

   *for each weight rule* $h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\}$ *in* $Q$.

The rules in $\mathrm{Least}^\bullet(\mathbb{Q})$ catch the least model $\mathrm{LM}(Q^N \cup \mathrm{F}_{N\cap I})$ for $N = M_\mathrm{v} \cup N_\mathrm{h}$ but expressed in $I \cup O^\bullet \cup \mathrm{Hb_h}(\mathbb{Q})^\circ$ rather than $\mathrm{Hb}(\mathbb{Q})$.

**Definition 6.4** *The translation*

$$\mathrm{UnStable}(\mathbb{Q}) = (\mathrm{UnStable}(Q), I \cup O \cup O^\bullet \cup \mathrm{Hb_h}(\mathbb{Q})^\bullet \cup \mathrm{Hb_h}(\mathbb{Q})^\circ, \{c, d, e\})$$

*of an* SMODELS *program module* $\mathbb{Q} = (Q, I, O)$ *includes*

1. *rules* $d \leftarrow a, {\sim}a^\bullet$ *and* $d \leftarrow a^\bullet, {\sim}a$ *for each* $a \in O$*;*
2. *rules* $d \leftarrow a^\circ, {\sim}a^\bullet$ *and* $d \leftarrow a^\bullet, {\sim}a^\circ$ *for each* $a \in \mathrm{Hb_h}(\mathbb{Q})$*;*
3. *a rule* $c \leftarrow {\sim}a^\bullet, {\sim}d$ *for each* $a \in \mathrm{CompS}(\mathbb{Q})$ *such that* $a \in O \cup \mathrm{Hb_h}(\mathbb{Q})$*;*
4. *a rule* $c \leftarrow b^\bullet, {\sim}d$ *for each* ${\sim}b \in \mathrm{CompS}(\mathbb{Q})$ *such that* $b \in O \cup \mathrm{Hb_h}(\mathbb{Q})$*;*
5. *a rule* $c \leftarrow {\sim}a, {\sim}d$ *for each* $a \in \mathrm{CompS}(\mathbb{Q})$ *such that* $a \in I$*;*
6. *a rule* $c \leftarrow b, {\sim}d$ *for each* ${\sim}b \in \mathrm{CompS}(\mathbb{Q})$ *such that* $b \in I$*;*
7. *rules* $e \leftarrow c$ *and* $e \leftarrow d$*; and*
8. *a compute statement* compute $\{e\}$.

The purpose of $\mathrm{UnStable}(\mathbb{Q})$ is to disqualify $N$ as a stable model of $\mathbb{Q}$: either $N$ and $\mathrm{LM}(Q^N \cup \mathrm{F}_{N\cap I})$ differ, or otherwise $\mathrm{LM}(Q^N \cup \mathrm{F}_{N\cap I})$ violates some compute statement of $\mathbb{Q}$. The rules in the second last item summarize the two possible reasons why $\mathbb{Q}$ does not have a stable model $N$ such that $M_\mathrm{v} = N_\mathrm{v}$. Atom $d$ is used to indicate that there is a difference between $N$ and $\mathrm{LM}(Q^N \cup \mathrm{F}_{N\cap I})$, and atom $c$ indicates that a compute statement of $\mathbb{Q}$ is violated. It is then insisted by the compute statement in the last item that either of these reasons holds.

Theorem 6.5 shows the correctness of the translation-based method for verification of modular equivalence.

**Theorem 6.5** *Let* $\mathbb{P} = (P, I, O)$ *and* $\mathbb{Q} = (Q, I, O)$ *be* SMODELS *program modules having enough visible atoms. Then* $\mathbb{P} \equiv_\mathrm{m} \mathbb{Q}$ *if and only if modules* $\mathrm{EQT}(\mathbb{P}, \mathbb{Q})$ *and* $\mathrm{EQT}(\mathbb{Q}, \mathbb{P})$ *have no stable models.*

Proof of Theorem 6.5 is given in Appendix A. Theorem 6.5 allows us to verify $\equiv_{\mathrm{m}}$ similarly to verifying $\equiv_{\mathrm{v}}$ in [31]. Notice that the generator module $\mathbb{G}_I$ from Section 4.2 or its variant consisting of a single choice rule, that is, module $(\{\{I\}.\}, \emptyset, I)$ can be combined with $\mathrm{EQT}(\mathbb{P}, \mathbb{Q})$ and $\mathrm{EQT}(\mathbb{Q}, \mathbb{P})$ in order to generate all the possible inputs in an actual implementation.

## 6.2 MODULAR APPROACH TO EQUIVALENCE VERIFICATION

Introduction of modular equivalence was much motivated by the need of modularizing the verification of weak/visible equivalence. We believe that such a modularization could be very effective in a setting where program $Q$ is an optimized version of program $P$ as this typically indicates that $Q$ is obtained by making some local modifications to $P$. In this section we propose a strategy to utilize modular equivalence in the task of verifying the visible/weak equivalence of SMODELS programs $P$ and $Q$. Since $\equiv_{\mathrm{m}}$ reduces to $\equiv_{\mathrm{v}}$ for programs with a completely specified input, the discussion will be given in the context of modules and $\equiv_{\mathrm{m}}$. The results established in the sequel are applicable to the cases of $\equiv_{\mathrm{v}}$ and $\equiv$ simply by restricting the input sets of the modules involved to be empty.

Consider SMODELS program modules $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$. To modularize further the verification of $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ one needs to have module structures of $\mathbb{P}$ and $\mathbb{Q}$. These can either be specified explicitly by the programmer or detected automatically, for example, by utilizing the strongly connected components of $\mathrm{Dep}^{+}(\mathbb{P})$ and $\mathrm{Dep}^{+}(\mathbb{Q})$. We will assume for now that modular structures for $\mathbb{P}$ and $\mathbb{Q}$ are given, and consider in Section 6.3 the question how to find suitable module structures for $\mathbb{P}$ and $\mathbb{Q}$ automatically in case that they are not known beforehand.

Given modules $\mathbb{P} = (P, I_P, O_P)$ and $\mathbb{Q} = (Q, I_Q, O_Q)$, we say that $\mathbb{P}$ is *compatible* with $\mathbb{Q}$, if $I_P = I_Q$ and $O_P = O_Q$. Now, if $\mathbb{Q}$ is obtained from $\mathbb{P}$ through local modifications, it is likely that their components are pairwise compatible, and there is a partitioning for $\mathbb{P}$ and $\mathbb{Q}$ such that $\mathbb{P} = \mathbb{P}_1 \sqcup \cdots \sqcup \mathbb{P}_n$ and $\mathbb{Q} = \mathbb{Q}_1 \sqcup \cdots \sqcup \mathbb{Q}_n$ where $\mathbb{P}_i$ is compatible with $\mathbb{Q}_i$ for all $i$. Notice that $\mathbb{P}_i = \mathbb{Q}_i$ might even hold for a number of $i$'s. On the other hand, it might be the case that $\mathbb{P}$ and $\mathbb{Q}$ have such dependencies that they have no non-trivial compatible module structures. This implies that the task of verifying the equivalence of $\mathbb{P}$ and $\mathbb{Q}$ cannot be further modularized.

Assume that $\mathbb{P} = \mathbb{P}_1 \sqcup \cdots \sqcup \mathbb{P}_n$ and $\mathbb{Q} = \mathbb{Q}_1 \sqcup \cdots \sqcup \mathbb{Q}_n$ are compatible module structures.[9] Verifying $\mathbb{P}_i \equiv_{\mathrm{m}} \mathbb{Q}_i$ for every $i$ is not of interest as such, since $\mathbb{P}_i \not\equiv_{\mathrm{m}} \mathbb{Q}_i$ does not necessarily imply $\mathbb{P} \not\equiv_{\mathrm{m}} \mathbb{Q}$. Of course, if $\mathbb{P}_i \equiv_{\mathrm{m}} \mathbb{Q}_i$ holds and the equivalence can be verified efficiently, then Corollary 5.19 implies that $\mathbb{P}_i$ and $\mathbb{Q}_i$ are modularly equivalent in every possible context. However, if this is not the case, it is still possible to organize the verification of $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$ as a sequence of $n$ module-level tests as follows:

$$(\bigsqcup_{j=1}^{i-1} \mathbb{Q}_j) \sqcup \mathbb{P}_i \sqcup (\bigsqcup_{j=i+i}^{n} \mathbb{P}_j) \equiv_{\mathrm{m}} (\bigsqcup_{j=1}^{i-1} \mathbb{Q}_j) \sqcup \mathbb{Q}_i \sqcup (\bigsqcup_{j=i+1}^{n} \mathbb{P}_j) \tag{6.1}$$

[9]Notice that several possible compatible module structures can be obtained for $\mathbb{P}$ and $\mathbb{Q}$, for example, by regrouping or taking compositions of submodules.

where $1 \leq i \leq n$. The resulting chain of equalities conveys $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$.

**Example 6.6** Consider normal logic programs $P$ and $Q$ both consisting of two submodules, that is, $P = \mathbb{P}_1 \sqcup \mathbb{P}_2$ and $Q = \mathbb{Q}_1 \sqcup \mathbb{Q}_2$, and the submodules are the following:

$$\begin{array}{rcl} \mathbb{P}_1 & = & (\{c \leftarrow \sim a.\}, \{a, b\}, \{c\}), \\ \mathbb{P}_2 & = & (\{a \leftarrow b.\}, \emptyset, \{a, b\}), \\ \mathbb{Q}_1 & = & (\{c \leftarrow \sim b.\}, \{a, b\}, \{c\}), \text{ and} \\ \mathbb{Q}_2 & = & (\{b \leftarrow a.\}, \emptyset, \{a, b\}). \end{array}$$

Now, $\mathbb{P}_1 \not\equiv_{\mathrm{m}} \mathbb{Q}_1$, but $\mathbb{P}_1$ and $\mathbb{Q}_1$ are modularly equivalent in contexts produced by both $\mathbb{P}_2$ and $\mathbb{Q}_2$. In this case actually $\mathbb{P}_2 \equiv_{\mathrm{m}} \mathbb{Q}_2$ holds as $\mathrm{SM}(\mathbb{P}_2) = \mathrm{SM}(\mathbb{Q}_2) = \{\emptyset\}$, but that is not necessary. Thus

$$\mathbb{P}_1 \sqcup \mathbb{P}_2 \equiv_{\mathrm{m}} \mathbb{Q}_1 \sqcup \mathbb{P}_2 \equiv_{\mathrm{m}} \mathbb{Q}_1 \sqcup \mathbb{Q}_2,$$

which verifies $P \equiv Q$ and $P \equiv_{\mathrm{v}} Q$. ■

The programs involved in each test (6.1) differ in $\mathbb{P}_i$ and $\mathbb{Q}_i$ for which the other modules form a common context

$$\mathbb{C}_i = (\bigsqcup_{j=1}^{i-1} \mathbb{Q}_j) \sqcup (\bigsqcup_{j=i+i}^{n} \mathbb{P}_j).$$

A way to optimize the verification of $\mathbb{P}_i \sqcup \mathbb{C}_i \equiv_{\mathrm{m}} \mathbb{Q}_i \sqcup \mathbb{C}_i$ is to adjust the method discussed in Section 6.1 to use translation $\mathrm{EQT}(\mathbb{P}_i, \mathbb{Q}_i) \sqcup \mathbb{C}_i$ rather than translation $\mathrm{EQT}(\mathbb{P}_i \sqcup \mathbb{C}_i, \mathbb{Q}_i \sqcup \mathbb{C}_i)$. The following theorem proves the correctness of this method.

**Theorem 6.7** *Let $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$ be* SMODELS *program modules with the EVA property, and $\mathbb{C}$ an* SMODELS *program module such that $\mathbb{P} \sqcup \mathbb{C}$ and $\mathbb{Q} \sqcup \mathbb{C}$ are defined. Then $\mathbb{P} \sqcup \mathbb{C} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{C}$ if and only if $\mathrm{EQT}(\mathbb{P}, \mathbb{Q}) \sqcup \mathbb{C}$ and $\mathrm{EQT}(\mathbb{Q}, \mathbb{P}) \sqcup \mathbb{C}$ have no stable models.*

Proof of Theorem 6.7 is given in Appendix A. Theorem 6.7 gives us the means to modularize the task of equivalence verification together with equation (6.1). Verifying $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$, when $\mathbb{P}$ and $\mathbb{Q}$ have compatible module structures $\mathbb{P} = \mathbb{P}_1 \sqcup \ldots \sqcup \mathbb{P}_n$ and $\mathbb{Q} = \mathbb{Q}_1 \sqcup \ldots \sqcup \mathbb{Q}_n$, is done as follows.

For each $1 \leq i \leq n$,

> if $\mathrm{SM}(\mathrm{EQT}(\mathbb{P}_i, \mathbb{Q}_i) \sqcup \mathbb{C}_i) \neq \emptyset$ or $\mathrm{SM}(\mathrm{EQT}(\mathbb{Q}_i, \mathbb{P}_i) \sqcup \mathbb{C}_i) \neq \emptyset$,
> break as $\mathbb{P} \not\equiv_{\mathrm{m}} \mathbb{Q}$, continue otherwise.

If no counterexample is found, then $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$. Note that if there are $n$ submodules, there are $n!$ possible different orders in which to verify the chain of equivalences. It seems likely that the order has an effect on the efficiency of the approach. We expect computational advantage from the strategy described above, especially when the context $\mathbb{C}_i$ is clearly larger than the modules $\mathbb{P}_i$ and $\mathbb{Q}_i$. This hypothesis is to be examined in the experimental part.

## 6.3 ON FINDING MODULE DECOMPOSITION

A prerequisite for the algorithm described in the previous section is to know compatible module structures for $\mathbb{P}$ and $\mathbb{Q}$. This is always not the case, and one needs to have a strategy how to find suitable module structures for the programs involved assuming that there is no a priori knowledge.

First idea for finding a module structure for SMODELS program module $\mathbb{P} = (P, I, O)$ is to use the strongly connected components $C_1, \ldots, C_n$ of $\mathrm{Dep}^+(\mathbb{P})$ and define submodules by grouping the rules so that for each $C_i$ all the rules $r \in P$ such that $\mathrm{Head}(r) \subseteq C_i$ are put into one submodule.

Now, the question is whether $\mathbb{P}_i$'s defined this way would form a valid decomposition of $\mathbb{P}$ into submodules. First notice that input atoms are a special case because $\mathrm{Head}(P) \cap I = \emptyset$. Each $a \in I$ ends up in its own strongly connected component and there are no rules to include into a submodule corresponding to strongly connected component $\{a\}$. Thus it is actually unnecessary to include a submodule based on such a component. Obviously, all basic rules, constraint rules and weight rules in $P$ go into exactly one of the submodules. One should notice that for a choice rule $r \in P$ it can happen that $\mathrm{Head}(r) \cap C_i \neq \emptyset$ and $\mathrm{Head}(r) \cap C_j \neq \emptyset$ for $i \neq j$. This is not a problem, since it is always possible to replace a choice rule of the form $\{H\} \leftarrow B^+, {\sim}B^-$ with choice rules $\{h\} \leftarrow B^+, {\sim}B^-$ for each $h \in H$.

In regard to compute statements in $\mathbb{P}$, there are several possibilities. One possibility is to put all of them in one submodule, which is then used as a submodule checking whether the conditions insisted upon stable models are satisfied by potential model candidates. Another approach is to place the compute statements into the submodule containing the definitions of the atoms which the compute statement concerns. We will use the latter strategy.[10]

Based on the discussion above, we define the *set of rules defining a set of atoms* for an SMODELS program module in analogy to the definition given in Section 2.2.2.

**Definition 6.8** *Given an* SMODELS *program module* $\mathbb{P} = (P, I, O)$ *and a set of atoms* $A \subseteq \mathrm{Hb}(\mathbb{P}) \setminus I$*, the set of rules defining* $A$*, denoted by* $P[A]$*, contains the following rules:*

- a basic rule $h \leftarrow B^+, {\sim}B^-$ if and only if there is a basic rule (5.1) in $P$ such that $h \in A$;
- a constraint rule $h \leftarrow c \leq \{B^+, {\sim}B^-\}$ if and only if there is a constraint rule (5.2) in $P$ such that $h \in A$;
- a choice rule $\{H \cap A\} \leftarrow B^+, {\sim}B^-$ if and only if there is a choice rule (5.3) in $P$ such that $H \cap A \neq \emptyset$;
- a weight rule $h \leftarrow c \leq \{B^+, {\sim}B^-\}$ if and only if there is a weight rule (5.4) in $P$ such that $h \in A$; and

[10] Note how LPARSE handles rules with an empty head. For example a basic rule of the form $\leftarrow B^+, {\sim}B^-$ is transformed to a rule $\bot \leftarrow B^+, {\sim}B^-$ and a compute statement compute $\{{\sim}\bot\}$ is added, where atom $\bot$ is a new hidden atom shared by all the rules transformed this way. Sharing the head atom groups all these rules together, and this resembles the first strategy in which compute statements are placed into one module.

- a compute statement compute $\{B^+ \cap A, {\sim}(B^- \cap A)\}$ if and only if there is a compute statement (5.5) in $P$ such that $B^+ \cap A \neq \emptyset$ or $B^- \cap A \neq \emptyset$.

We continue by defining a submodule of $\mathbb{P} = (P, I, O)$ induced by a set of atoms $A \subseteq \mathrm{Hb}(\mathbb{P}) \setminus I$. We use Definition 6.8 for the set of rules, and choose $A_\mathrm{v}$ to be the output and the rest of the visible atoms appearing in $P[A]$ to be the input. Let $\mathrm{atoms}_\mathrm{v}(P)$ denote the set of visible atoms appearing in the rules of $P$.

**Definition 6.9** *Given an* SMODELS *program module* $\mathbb{P} = (P, I, O)$ *and a set of atoms* $A \subseteq \mathrm{Hb}(\mathbb{P}) \setminus I$*, a submodule induced by* $A$ *is*

$$\mathbb{P}[A] = (P[A], \mathrm{atoms}_\mathrm{v}(P[A]) \setminus A_\mathrm{v}, A_\mathrm{v}).$$

Let $C_1, \ldots C_m$ be the strongly connected components of $\mathrm{Dep}^+(\mathbb{P})$ such that $C_i \cap I = \emptyset$. Now we can define $\mathbb{P}_i = \mathbb{P}[C_i]$ for each $1 \leq i \leq m$. Since the strongly connected components of $\mathrm{Dep}^+(\mathbb{P})$ are used as a basis, it is guaranteed that there is no positive recursion between any of the submodules $\mathbb{P}_i$. Also, it is clear that outputs of submodules are pairwise disjoint. Unfortunately this construction does not yet guarantee that hidden atoms stay local, and therefore the composition of $\mathbb{P}_i$'s might not be defined.

A solution is to combine $C_i$'s in a way that modules will be closed with respect to dependencies on the hidden atoms, that is, if a hidden atom $h$ belongs to a component $C_i$, then also all the atoms in the heads of rules in which $h$ or ${\sim}h$ appears, have to belong to $C_i$, too. This can be achieved by finding the strongly connected components, denoted by $D_1, \ldots, D_k$, for $\mathrm{Dep}^\mathrm{h}(\mathbb{P}, \{C_1, \ldots, C_m\})$, where $\mathrm{Dep}^\mathrm{h}(\mathbb{P}, \{C_1, \ldots, C_m\})$ has $\{C_1, \ldots, C_m\}$ as the set of vertices, and

$$\{\langle C_i, C_j \rangle, \langle C_j, C_i \rangle \mid a \in C_i, b \in C_j, \\ r \in P, b \in \mathrm{Head}(r) \text{ and } a \in (\mathrm{Body}^+(r) \cup \mathrm{Body}^-(r))_\mathrm{h}\}$$

as the set of edges. Now, we take the sets

$$E_i = \bigcup_{C \in D_i} C$$

for $1 \leq i \leq k$ and use them to induce a module structure for $\mathbb{P}$ by defining $\mathbb{P}_i = \mathbb{P}[E_i]$ for $1 \leq i \leq k$.

As it is possible to have atoms in $I \cup O$ not appearing in the rules of $\mathbb{P}$, that is, $\mathrm{Hb}(\mathbb{P}) = \mathrm{atoms}_\mathrm{v}(P)$ does not necessarily hold, it is possible that

$$\mathrm{Hb}(\mathbb{P}) \setminus (\mathrm{Hb}(\mathbb{P}_1) \cup \cdots \cup \mathrm{Hb}(\mathbb{P}_k)) \neq \emptyset.$$

To keep track of such atoms in $I \setminus \mathrm{atoms}_\mathrm{v}(P)$ we need an additional module defined as

$$\mathbb{P}_0 = (\emptyset, I \setminus \mathrm{atoms}_\mathrm{v}(P), \emptyset).$$

Note that there is no need for similar treatment for atoms in $O \setminus \mathrm{atoms}_\mathrm{v}(P)$ as each atom in $O$ belongs to some $\mathrm{Hb}(\mathbb{P}_i)$.

Theorem 6.10 shows that we have a valid decomposition of $\mathbb{P}$ into submodules.

**Theorem 6.10** *Consider an* SMODELS *logic program module* $\mathbb{P}$ *and let* $C_1, \ldots C_m$ *be the SCCs of* $\mathrm{Dep}^{+}(\mathbb{P})$ *such that* $C_i \cap I = \emptyset$*, and* $D_1, \ldots, D_k$ *the SCCs of* $\mathrm{Dep}^{\mathrm{h}}(\mathbb{P}, \{C_1, \ldots, C_k\})$*. Define* $E_i = \cup_{C \in D_i} C$*,* $\mathbb{P}_i = \mathbb{P}[E_i]$*, and* $\mathbb{P}_0 = (\emptyset, I \setminus \mathrm{atoms}_{\mathrm{v}}(P), \emptyset)$*. Then the join of the submodules* $\mathbb{P}_i$ *is defined and* $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{P}_0 \sqcup \cdots \sqcup \mathbb{P}_m$*.*

**Proof sketch for Theorem 6.10** Based on the construction of $E_i$'s and the discussion in this section it is clear that $\mathbb{P}' = \mathbb{P}_0 \sqcup \cdots \sqcup \mathbb{P}_m$ is defined. It is easy to verify that the input and the output sets of $\mathbb{P}'$ and $\mathbb{P}$ are exactly the same. The only difference between the sets of rules in $\mathbb{P}$ and $\mathbb{P}'$ is that some choice rules and compute statements in $\mathbb{P}$ may be split into several rules in $\mathbb{P}'$. This is merely a syntactical change not affecting the stable models of the modules, that is, $\mathrm{SM}(\mathbb{P}) = \mathrm{SM}(\mathbb{P}')$. Notice also that $\mathrm{Dep}^{+}(\mathbb{P}) = \mathrm{Dep}^{+}(\mathbb{P}')$. Thus $\mathbb{P}' \equiv_{\mathrm{m}} \mathbb{P}$ holds. □

Considering the problem of modularizing the verification of $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$, it is possible to obtain module structures for $\mathbb{P}$ and $\mathbb{Q}$ separately using the approach discussed above. However, this does not necessarily yield compatible module structures for $\mathbb{P}$ and $\mathbb{Q}$, and a further step is needed to achieve compatibility.

An approach to find directly compatible module structures for SMODELS program modules $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$ is to take $\mathrm{Dep}^{+}(P \cup Q)$ as a starting point, and proceed to find the set of strongly connected components of $\mathrm{Dep}^{+}(P \cup Q)$ closed with respect to hidden atoms as done before. We denote these by $E_1, \ldots, E_k$. Compatible submodules $\mathbb{P}_i = \mathbb{P}[E_i]$ and $\mathbb{Q}_i = \mathbb{Q}[E_i]$ for $1 \leq i \leq k$ can be defined according to Definition 6.9 with the exception, that the input needs to be defined as

$$\mathrm{atoms}_{\mathrm{v}}((P \cup Q)[E_i]) \setminus (E_i)_{\mathrm{v}}.$$

Similarly we need to define $\mathbb{P}_0 = \mathbb{Q}_0 = (\emptyset, I \setminus \mathrm{atoms}_{\mathrm{v}}(P \cup Q), \emptyset)$.

Note that it might be possible to find further compatible decompositions for some submodules $\mathbb{P}_i$ and $\mathbb{Q}_i$, that is, the approach described above does not guarantee that the compatible decompositions into submodules obtained for $\mathbb{P}$ and $\mathbb{Q}$ are as fine as possible. However, this is not important as such. Having a decomposition into too many or too small submodules might actually turn out to be inefficient in practice.

# 7 EXPERIMENTS

In this chapter we evaluate experimentally the effect of modularization on verification of visible equivalence compared to a non-modular approach. We start by introducing a prototype implementation and practical arrangements in our experiments in Section 7.1. The problem of placing $n$ queens on a $n \times n$ chess board is used as our first benchmark and results are reported in Section 7.2. Results for the second benchmark, the problem of finding Hamiltonian cycles for directed graphs of $n$ vertices, are reported in Section 7.3.

## 7.1 TEST ARRANGEMENTS

The translation function EQT for verifying modular equivalence presented in Definition 6.1 has been incorporated to translator LPEQ (version 1.18)[11] by Docent Tomi Janhunen. The translation for verifying modular equivalence is obtained using flag `-m`. Furthermore, to make the translation compatible with current solvers, an input generator can be augmented to the translation using flag `-i`. Together with a tool called LPCAT (version 1.6) used to compose two modules together, the new version of LPEQ allows us to examine the feasibility of verification of modular equivalence, compared to the approach where programs are seen as integral entities.

It is worth noticing that even though LPEQ and LPCAT give us the means to evaluate the modular approach to equivalence verification, current versions of LPARSE and SMODELS are not yet fully compatible with the concept of modules. This brings some difficulties, which we have tried to overcome. The following example illustrates how LPEQ and LPCAT can be used together with LPARSE and SMODELS.

**Example 7.1** Consider programs $P$ and $Q$ consisting of modules $\mathbb{P}_1$, $\mathbb{P}_2$, $\mathbb{Q}_1$ and $\mathbb{Q}_2$ given in Example 6.6. Their encoding in the input language of LPARSE is given in Figure 7.1.

Declaration `#external` is used to tell LPARSE which atoms are not defined inside the module. This declaration does not match exactly what we would like the declaration of input atoms to be, but currently there is no direct support for declaring input atoms in LPARSE. Information hiding, however, can be obtained using `#hide` declaration. For more details see the user's manual of LPARSE [67].

LPARSE can be used to obtain the presentation of the modules in the internal format of SMODELS which is also the input format for LPEQ and LPCAT. For example, in the case of module $\mathbb{P}_1$:

```
$ lparse p1.lp > p1.sm
3: Warning: predicate 'a/0' doesn't occur in any rule
   head.
$ more p1.sm
1 2 1 1 3
```

[11]Available at `http://www.tcs.hut.fi/Software/lpeq/`.

```
p1.lp:   #external a, b.        p2.lp:   #external b.
         c :- not a.                     a :- b.

q1.lp:   #external a,b.         q2.lp:   #external a.
         c :- not b.                     b :- a.
```

Figure 7.1: LPARSE encodings of modules $\mathbb{P}_1$, $\mathbb{P}_2$, $\mathbb{Q}_1$ and $\mathbb{P}_2$ given in Example 7.1.

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

Since LPARSE is designed for programs with a completely specified input, there is a warning message. Notice also that since atom $b$ does not occur in the rules of $\mathbb{P}_1$, it does not appear in the symbol table of `p1.sm`. It is possible to add $b$ to the symbol table by, for example, inserting a line "`4 b`" to the symbol table in `p1.sm`. Files `p2.sm`, `q1.sm` and `q2.sm` can be obtained similarly from files `p2.lp`, `q1.lp` and `q2.lp`, respectively.

LPCAT computes the join of two modules with flag `-m`. For example, $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is obtained as follows.

```
$ lpcat -m p1.sm p2.sm > p1-p2.sm
```

The composition `q2-p2.sm` can be obtained similarly.

Now, we are ready to use LPEQ to verify the (visible) equivalence of $P$ and $Q$. First, we can verify $P \equiv_{\mathrm{v}} Q$ without taking into account our knowledge about the modular structure using the following commands.

```
$ lpeq p1-p2.sm q1-q2.sm | smodels 1
$ lpeq q1-q2.sm p1-p2.sm | smodels 1
```

The modular approach to equivalence verification goes as follows. In the first step one computes the translation $\mathrm{EQT}(\mathbb{P}_1, \mathbb{Q}_1)$ using LPEQ with flag `-m`. The context $\mathbb{P}_2$ is added using the tool LPCAT with flag `-m` and SMODELS is used to see if the translation $\mathrm{EQT}(\mathbb{P}_1, \mathbb{Q}_1) \sqcup \mathbb{P}_2$ has stable models.

```
$ lpeq -m p1.sm q1.sm | lpcat -m - p2.sm | smodels 1
```

Second direction is verified in the same way.

```
$ lpeq -m q1.sm p1.sm | lpcat -m - p2.sm | smodels 1
```

Since neither translation has stable models we learn that $\mathbb{P}_1 \sqcup \mathbb{P}_2 \equiv_{\mathrm{m}} \mathbb{Q}_1 \sqcup \mathbb{P}_2$. The second equivalence in the chain, that is, $\mathbb{Q}_1 \sqcup \mathbb{P}_2 \equiv_{\mathrm{m}} \mathbb{Q}_1 \sqcup \mathbb{Q}_2$, is verified similarly:

```
$ lpeq -m p2.sm q2.sm | lpcat -m - q1.sm | smodels 1
$ lpeq -m q2.sm p2.sm | lpcat -m - q1.sm | smodels 1
```

It is also possible to use a different order in the chaining and verify the chain of equivalences $\mathbb{P}_1 \sqcup \mathbb{P}_2 \equiv_{\mathrm{m}} \mathbb{P}_1 \sqcup \mathbb{Q}_2 \equiv_{\mathrm{m}} \mathbb{Q}_1 \sqcup \mathbb{Q}_2$.

Even though $\mathbb{P}_1$ and $\mathbb{Q}_1$ behave equivalently in the contexts of $\mathbb{P}_2$ and $\mathbb{Q}_2$, they are not modularly equivalent in general. This can be verified using the translation without a specific context. Note that the translation $\mathrm{EQT}(\mathbb{P}_1, \mathbb{Q}_1)$ is a module with input $\{a, b\}$. To be able to compute its stable models using the current SMODELS one needs to attach an input generator. This can be obtained automatically using flag `-i` in LPEQ.

```
$ lpeq -m -i p1.sm q1.sm | smodels 1
smodels version 2.32. Reading...done
Answer: 1
Stable Model: a c'
True
Duration: 0.002
Number of choice points: 1
Number of wrong choices: 0
Number of atoms: 9
Number of rules: 8
Number of picked atoms: 3
Number of forced atoms: 0
Number of truth assignments: 13
Size of searchspace (removed): 2 (2)
```

The stable model $\{a, c^\bullet\}$ for translation $\mathrm{EQT}(\mathbb{P}_1, \mathbb{Q}_1)$ gives us a counterexample: $\{a\} \in \mathrm{SM}(\mathbb{P}_1)$ and $\{a, c\} = \mathrm{LM}(\mathbb{Q}_1{}^{\{a\}} \cup \mathrm{F}_{\{a,b\}\cap\{a\}})$, thus we have $\{a\} \notin \mathrm{SM}(\mathbb{Q}_1)$. ■

In our experiments SMODELS system (version 2.32) is responsible for the computation of stable models with flag `-nolookahead` (for the benchmarks in this report, this option turned out to give the shortest running times) for programs that are instantiated using the front-end LPARSE (version 1.0.17). The total running time for the equivalence verification in one direction is the sum of running times needed by SMODELS for trying to compute *one* stable model of each of the translations produced by LPEQ. Recall that in the modular approach there is a chain of equivalences needed to verify. The translation time is also taken into account although it is negligible. We consider modularly equivalent programs (modules) and therefore one always has to count running times in both directions. Note, however, that running times scale differently depending on the direction. Since the running times of SMODELS depend on the order of rules in programs and literals in rules, we shuffle them randomly. All the tests reported were run under the Linux 2.6.8 operating system on a 1.7GHz AMD Athlon XP 2000+ CPU with 1 GB of main memory. As regards timings in test results, we report the sum of user and system times as measured by `/usr/bin/time` command in UNIX.

## 7.2 THE QUEENS BENCHMARK

We consider modular encodings solving the $n$-queens problem, that is, how to place $n$ queens on a $n \times n$ chess board so that they do not threaten each other. The programs are composed of two modules each, where the first module is either $\mathbb{G}_{\mathrm{x}}^n$ or $\mathbb{G}_{\mathrm{y}}^n$ and the second module is either $\mathbb{C}_1^n$ or $\mathbb{C}_2^n$. Basically $\mathbb{G}_{\mathrm{x}}^n$ and $\mathbb{G}_{\mathrm{y}}^n$ generate a placement of $n$ queens row-by-row and column-by-column, respectively. The input sets of generator modules $\mathbb{G}_{\mathrm{x}}^n$ and $\mathbb{G}_{\mathrm{y}}^n$ are empty, and their output consists of ground instances of predicate `q(X,Y)`. Modules $\mathbb{C}_1^n$ and $\mathbb{C}_2^n$ are used to check that the placement induced by the generator module is legal. The check modules $\mathbb{C}_1^n$ and $\mathbb{C}_2^n$ have empty output, and their input consists of ground instances of predicate `q(X,Y)`. All four possible compositions of modules are

$$\begin{aligned} Q_1^n &= \mathbb{G}_{\mathrm{x}}^n \sqcup \mathbb{C}_1^n, \\ Q_2^n &= \mathbb{G}_{\mathrm{x}}^n \sqcup \mathbb{C}_2^n, \\ Q_3^n &= \mathbb{G}_{\mathrm{y}}^n \sqcup \mathbb{C}_1^n, \text{ and} \\ Q_4^n &= \mathbb{G}_{\mathrm{y}}^n \sqcup \mathbb{C}_2^n. \end{aligned}$$

We are mainly interested in the visible equivalence of $Q_1^n$ and $Q_4^n$, whereas $Q_2^n$ and $Q_3^n$ serve as (alternative) intermediates steps in the chain (6.1) used to modularize the verification of visible equivalence.

The encodings of $\mathbb{C}_1^n$, $\mathbb{C}_2^n$, $\mathbb{G}_{\mathrm{x}}^n$ and $\mathbb{G}_{\mathrm{y}}^n$ are given in Appendix B.1. Note that the encodings contain variables and need to be grounded using LPARSE to obtain propositional modules. Predicate `d(X)` appearing in the encodings is a domain predicate used in order to get an instantiation for `q(X,Y)`, and `q` is a constant denoting the parameter $n$, that is, the number of queens. Since LPARSE is basically intended for programs with completely specified input, we need to define predicate `q(X,Y)` external in modules $\mathbb{C}_1$ and $\mathbb{C}_2$. The flag `-d none` of LPARSE is used to remove domain predicates from the output. For example, ground instances of modules $\mathbb{G}_{\mathrm{x}}^2$ and $\mathbb{C}_2^2$ are given in Figure 7.2 in the textual format provided by LPARSE.

First, we compare the time needed to verify

(i) $\mathbb{C}_1^n \equiv_{\mathrm{m}} \mathbb{C}_2^n$,

(ii) $\mathbb{C}_1^n \sqcup \mathbb{G}_{\mathrm{x}}^n \equiv_{\mathrm{m}} \mathbb{C}_2^n \sqcup \mathbb{G}_{\mathrm{x}}^n$, and

(iii) $Q_1^n \equiv_{\mathrm{v}} Q_2^n$.

Notice that in (ii) and (iii) exactly the same equivalence is verified, the difference being that in (iii) the knowledge about modular structure is not taken into account and the translation-based method is applied to complete programs. In (ii) we use the approach described in Theorem 6.7 with $\mathbb{G}_{\mathrm{x}}^n$ as the common context for $\mathbb{C}_1^n$ and $\mathbb{C}_2^n$. In (i) we verify the equivalence of $\mathbb{C}_1^n$ and $\mathbb{C}_2^n$ in every possible context. This is a stronger result than what is obtained in (ii) and (iii). By Theorem 4.2 it is clear that equivalence in (i) implies equivalence (ii) and (iii), but not vice versa.

The number of queens $n$ was varied from 4 to 11 and the verification task was repeated 10 times for each number of queens generating each time

```
$ lparse -c q=2 -t gen-x.lp
_false :- __int2(_,1).
__int2(_,1) :- 2 { not q(1,2), not q(1,1) }.
_false :- __int2(_,2).
__int2(_,2) :- 2 { not q(2,2), not q(2,1) }.
_false :- __int1(_,1).
__int1(_,1) :- 2 {  q(1,2),  q(1,1) }.
_false :- __int1(_,2).
__int1(_,2) :- 2 {  q(2,2),  q(2,1) }.
{ q(1,2), q(1,1) }.
{ q(2,2), q(2,1) }.
compute 1 { not _false }.

$ lparse -c q=2 -t check-2.lp
_false :- q(2,1), q(1,1).
_false :- q(2,2), q(1,2).
_false :- q(1,2), q(1,1).
_false :- q(2,2), q(2,1).
_false :- q(1,2), q(2,1).
_false :- q(1,1), q(2,2).
compute 1 { not _false }.
```

Figure 7.2: Modules $\mathbb{G}^2_{\mathrm{x}}$ and $\mathbb{C}^2_2$ in grounded by LPARSE.

new randomly shuffled versions of the modules involved. Notice that after grounding the number of rules in $\mathbb{G}^n_{\mathrm{x}}$ and $\mathbb{G}^n_{\mathrm{y}}$ is linear to $n$ (for example, $\mathbb{G}^n_{\mathrm{x}}$ has 20, 30, and 40 rules for $n = 4, 6$, and 8, respectively) whereas the number of rules in $\mathbb{C}^n_1$ and $\mathbb{C}^n_2$ is of order $n^3$ (for example, $\mathbb{C}^n_1$ has 152, 580, and 1456 rules for $n = 4, 6$, and 8, respectively; and $\mathbb{C}^n_2$ has 76, 290, and 728 rules for $n = 4, 6$, and 8, respectively).

The average running times and the average numbers of choice points, that is, the number of choices made by SMODELS during the search, for each approach are presented in Figure 7.3. We see that taking into account the common context improves the efficiency of the translation-based method, as regards both time and the number of choice points. Checking modular equivalence without a specific context can be time consuming if the number

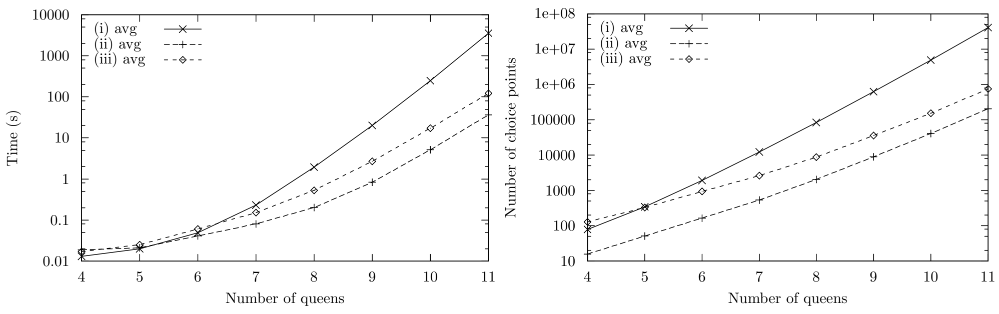

Figure 7.3: The average running times (left) and the average numbers of choice points (right) for verifying (i) $\mathbb{C}^n_1 \equiv_{\mathrm{m}} \mathbb{C}^n_2$, (ii) $\mathbb{C}^n_1 \sqcup \mathbb{G}^n_{\mathrm{x}} \equiv_{\mathrm{m}} \mathbb{C}^n_2 \sqcup \mathbb{G}^n_{\mathrm{x}}$, and (iii) $Q^n_1 \equiv_{\mathrm{v}} Q^n_2$ in the first $n$-queens experiment.

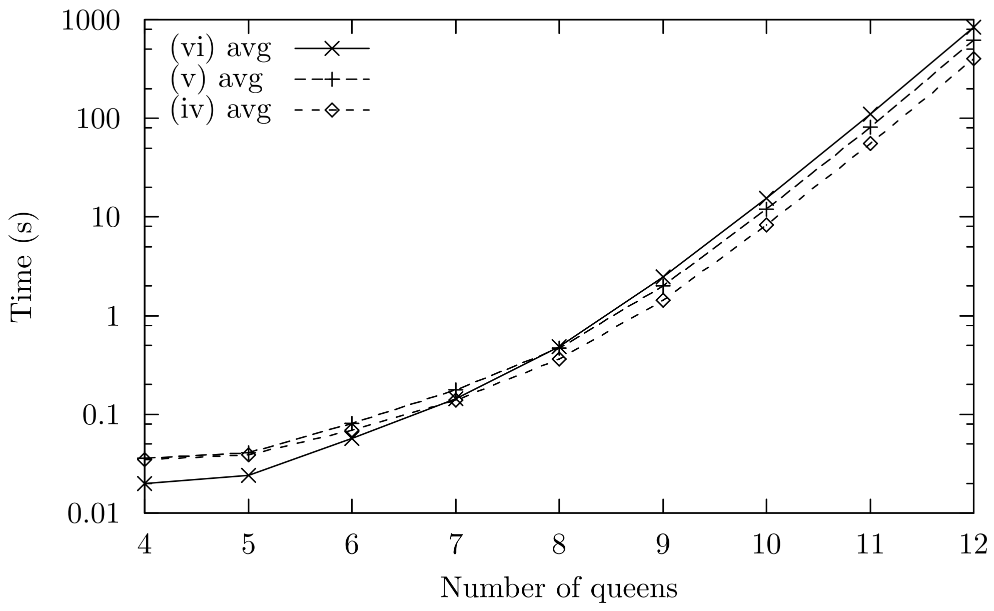

Figure 7.4: The average running times for verifying (iv) $\mathbb{C}_1^n \sqcup \mathbb{G}_x^n \equiv_m \mathbb{C}_2^n \sqcup \mathbb{G}_x^n \equiv_m \mathbb{C}_2^n \sqcup \mathbb{G}_y^n$, (v) $\mathbb{C}_1^n \sqcup \mathbb{G}_x^n \equiv_m \mathbb{C}_1^n \sqcup \mathbb{G}_y^n \equiv_m \mathbb{C}_2^n \sqcup \mathbb{G}_y^n$, and (vi) $Q_1^n \equiv_v Q_4^n$ in the second $n$-queens experiment.

of possible inputs is high, as is the case with the queens encodings (there are $n^2$ squares in the chess board and as a queen can either be placed or not in each of them, there are $2^{n^2}$ possible inputs for modules $\mathbb{C}_1^n$ and $\mathbb{C}_2^n$). However, it should be kept in mind that once $\mathbb{C}_1^n \equiv_m \mathbb{C}_2^n$ has been verified, one knows that $\mathbb{C}_1^n$ and $\mathbb{C}_2^n$ are modularly equivalent in any possible context.

In our second experiment, we compare the time needed to verify

(iv) $\mathbb{C}_1^n \sqcup \mathbb{G}_x^n \equiv_m \mathbb{C}_2^n \sqcup \mathbb{G}_x^n$ plus $\mathbb{C}_2^n \sqcup \mathbb{G}_x^n \equiv_m \mathbb{C}_2^n \sqcup \mathbb{G}_y^n$;

(v) $\mathbb{C}_1^n \sqcup \mathbb{G}_x^n \equiv_m \mathbb{C}_1^n \sqcup \mathbb{G}_y^n$ plus $\mathbb{C}_1^n \sqcup \mathbb{G}_y^n \equiv_m \mathbb{C}_2^n \sqcup \mathbb{G}_y^n$; and

(vi) $Q_1^n \equiv_v Q_4^n$.

In each item exactly the same equivalence is verified. The purpose of this experiment is to evaluate the efficiency of the modular method, if there are local changes in more modules, that is, when one verifies a chain of equivalence justifying the modular changes. Again we compare this strategy to the non-modular one. Another aspect is to see whether the order of the equivalences in the chain has an effect on the efficiency, that is, whether there is a difference in running times between items (iv) and (v).

In this experiment the number of queens $n$ was varied from 4 to 12 and the verification task was repeated 10 times for each number of queens generating each time new randomly shuffled versions of the modules involved. The average running times for each approach are presented in Figure 7.4. Again, the approaches that take modularity into account are slightly faster than the one that treats programs as integral entities. The order in which the chain of equivalences is evaluated has an effect of the running times, but not on the number of choice points as the average numbers of choice points in each approach was approximately the same.

## 7.3 THE HAMILTONIAN CYCLE BENCHMARK

We use the problem of finding a Hamiltonian cycle for directed graphs of $n$ vertices as our second benchmark problem. Basically, we consider two modularly equivalent encodings of the Hamiltonian cycle problem.

The first encoding consists of three modules $\mathbb{G}_1^n \sqcup \mathbb{H}_1^n \sqcup \mathbb{R}^n$, and is similar to the Hamiltonian cycle encoding used by Simons et al. [65]. We, however, consider directed graphs instead of undirected ones. Module $\mathbb{G}_1^n$ is used to generate all possible directed graphs of $n$ vertices represented as a set of edges. Given a set of edges for $n$ vertices as an input, module $\mathbb{H}_1^n$ selects the edges to be taken into a cycle by insisting that each vertex is incident to exactly two edges in the cycle. Finally, given a cycle candidate as an input, module $\mathbb{R}^n$ checks that each vertex is reachable from the starting vertex along the edges in the cycle. We also use an optimized variant of module $\mathbb{H}_1^n$. Module $\mathbb{H}_2^n$ takes into account that the input graph is directed and each vertex must then have exactly one incoming and exactly one outgoing edge in the cycle.

The second encoding $\mathbb{G}_1^n \sqcup \mathbb{HR}^n$ is based on the alternative encoding presented in [65]. In this encoding we cannot separate the selection of the edges to be taken into the cycle and the checking of reached vertices into two modules as their definitions are in positive recursion. Thus module $\mathbb{HR}^n$ solves the Hamiltonian cycle problem given a graph of $n$ vertices as input.

In addition to the module $\mathbb{G}_1^n$ generating all directed graphs, we consider also other graph generator modules. Each module $\mathbb{G}_i^n$ for $i = 1, \ldots, 5$ generates a family of directed graphs with $n$ vertices with the following properties:

- $\mathbb{G}_1^n$: all (directed) graphs,
- $\mathbb{G}_2^n$: irreflexive graphs,
- $\mathbb{G}_3^n$: symmetric and irreflexive graphs,
- $\mathbb{G}_4^n$: asymmetric graphs, and
- $\mathbb{G}_5^n$: graphs with Euclidean edge relation.

Encodings for modules $\mathbb{G}_i^n$ $(i = 1, \ldots, 5)$, $\mathbb{R}^n$, $\mathbb{H}_j^n$ $(j = 1, 2)$ and $\mathbb{HR}^n$ in the input language of LPARSE are given in Appendix B.2. In the experiments with the Hamiltonian cycle problem the number of vertices $n$ was varied from 3 to 10 and the verification task was repeated 10 times for each number of vertices generating each time new randomly shuffled versions of the modules involved.

First, we compare the time needed to verify

(a) $\mathbb{H}_1^n \sqcup (\mathbb{R}^n \sqcup \mathbb{G}_1^n) \equiv_{\mathrm{m}} \mathbb{H}_2^n \sqcup (\mathbb{R}^n \sqcup \mathbb{G}_1^n)$,

(b) $(\mathbb{H}_1^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_1^n \equiv_{\mathrm{m}} (\mathbb{H}_2^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_1^n$, and

(c) $\mathbb{H}_1^n \sqcup \mathbb{R}^n \sqcup \mathbb{G}_1^n \equiv_{\mathrm{m}} \mathbb{H}_2^n \sqcup \mathbb{R}^n \sqcup \mathbb{G}_1^n$.

In this experiment we want to see what effect varying the size of the common context has on verification of equivalence of $\mathbb{H}_1^n \sqcup \mathbb{R}^n \sqcup \mathbb{G}_1^n$ and $\mathbb{H}_2^n \sqcup \mathbb{R}^n \sqcup \mathbb{G}_1^n$. Thus the equivalence verified in each item is the same and the difference

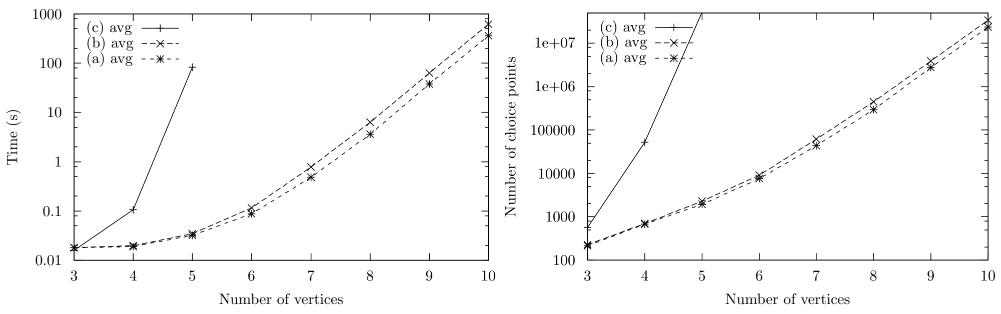

Figure 7.5: The average running times (left) and the average numbers of choice points (right) for verifying (a) $\mathbb{H}_1^n \sqcup (\mathbb{R}^n \sqcup \mathbb{G}_1^n) \equiv_{\mathrm{m}} \mathbb{H}_2^n \sqcup (\mathbb{R}^n \sqcup \mathbb{G}_1^n)$, (b) $(\mathbb{H}_1^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_1^n \equiv_{\mathrm{m}} (\mathbb{H}_2^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_1^n$, and (c) $\mathbb{H}_1^n \sqcup \mathbb{R}^n \sqcup \mathbb{G}_1^n \equiv_{\mathrm{m}} \mathbb{H}_2^n \sqcup \mathbb{R}^n \sqcup \mathbb{G}_1^n$ in the first Hamiltonian cycle experiment.

is that the common context is varied from $\mathbb{R}^n \sqcup \mathbb{G}_1^n$ in item (a) to empty (module) in item (c). The average running times and the average numbers of choice points for the approaches are presented in Figure 7.5.

We see that the approach in item (c) becomes infeasible for graphs with only six vertices (we used a timeout of 6000 seconds here). The difference in running times for equivalences in items (a) and (b) is smaller, but best results are achieved when the maximal common context is used in (a). The average number of choices made by SMODELS behave similarly to the average running times. For comparison, the time needed to find all stable models for $\mathbb{H}_2^n \sqcup \mathbb{R}^n \sqcup \mathbb{G}_1^n$ is approximately 1.5 seconds for $n = 4$ and 2000 seconds for $n = 5$. This shows the effectiveness of the modular translation-based method as a naive approach cross-checking the stable models is likely to be infeasible even for $n = 5$, because the number of stable models becomes very high.

Next we compare the time needed to verify

(i) equivalence in item (a) and $(\mathbb{H}_2^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_1^n \equiv_{\mathrm{m}} \mathbb{HR}^n \sqcup \mathbb{G}_1^n$, and

(ii) $(\mathbb{H}_1^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_1^n \equiv_{\mathrm{m}} \mathbb{HR}^n \sqcup \mathbb{G}_1^n$.

Notice that the equivalence verified in item (i) is the same as verified in item (ii). The motivation is to see whether it is more efficient to verify the equivalence of $\mathbb{H}_1^n \sqcup \mathbb{R}^n \sqcup \mathbb{G}_1^n$ and $\mathbb{HR}^n \sqcup \mathbb{G}_1^n$ directly or having $\mathbb{H}_2^n \sqcup \mathbb{R}^n \sqcup \mathbb{G}_1^n$ as an intermediate step. Furthermore, we wanted to see how the time needed to verify modular equivalence changes, if we hide predicate `reached(X)` in the encodings. Recall that this increases the size of the translation. Thus, we verify the equivalences (i) and (ii) also with predicate `reached(X)` hidden and compare the time needed when predicate `reached(X)` is visible.

The average running times for this experiment are presented in Figure 7.6 and the average numbers of choice points in Figure 7.7. Hiding predicate `reached(X)` in approach (i) increases the average running time by approximately one third. The effect of hiding is more significant in approach (ii) in which the average running time is doubled when predicate `reached(X)` is hidden. Thus, in practice it seems to be a good idea to hide as few predicates as necessary. The difference in the average numbers of choice points is marginal.

To answer the question whether it is more efficient to first replace $\mathbb{H}_1^n$ by $\mathbb{H}_2^n$ and verify modular equivalence of this step in context $\mathbb{R}^n \sqcup \mathbb{G}_1^n$ and then

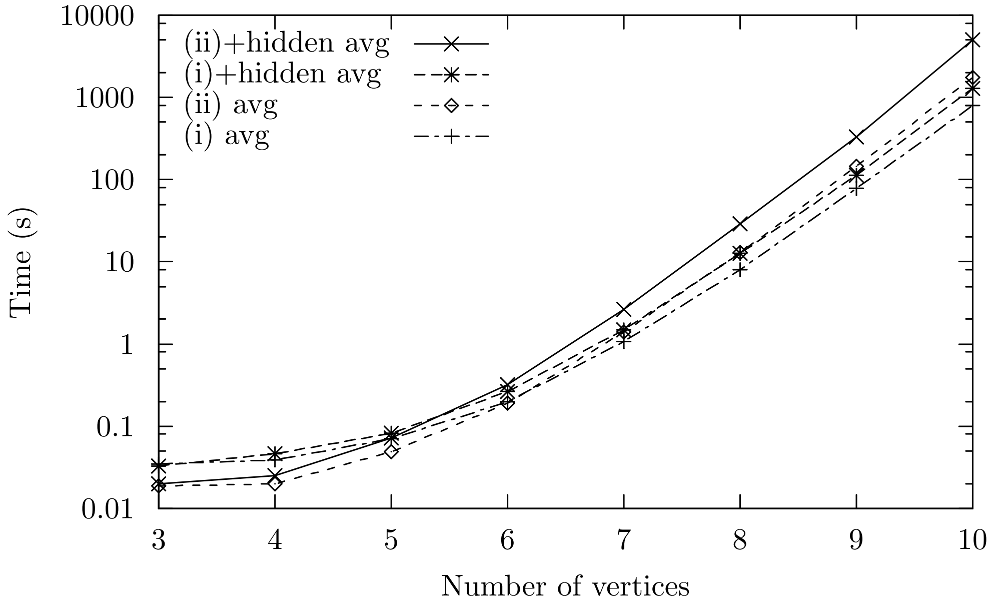

Figure 7.6: The average running times for verifying (i) $\mathbb{H}_1^n \sqcup (\mathbb{R}^n \sqcup \mathbb{G}_1^n) \equiv_{\mathrm{m}} \mathbb{H}_2^n \sqcup (\mathbb{R}^n \sqcup \mathbb{G}_1^n)$ plus $(\mathbb{H}_2^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_1^n \equiv_{\mathrm{m}} \mathbb{HR}^n \sqcup \mathbb{G}_1^n$, and (ii) $(\mathbb{H}_1^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_1^n \equiv_{\mathrm{m}} \mathbb{HR}^n \sqcup \mathbb{G}_1^n$ with predicate `reached(X)` hidden/visible in the second Hamiltonian cycle experiment.

verify $(\mathbb{H}_2^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_1^n \equiv_{\mathrm{m}} \mathbb{HR}^n \sqcup \mathbb{G}_1^n$; or to verify $(\mathbb{H}_1^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_1^n \equiv_{\mathrm{m}} \mathbb{HR}^n \sqcup \mathbb{G}_1^n$ directly, it seems that it is a good idea to do the optimization step first. The average running times of approach (i) are less than those of approach (ii) regardless the visibility of predicate `reached(X)`. The average numbers of choice points are somewhat higher in approach (i) than in (ii).

Finally, we wanted to see how the choice of the graph family, that is, the choice of the graph generator module $\mathbb{G}_i^n$ for $i = 1, 2, \ldots, 5$ affects the time needed to verify

(iii) $\mathbb{H}_1^n \sqcup (\mathbb{R}^n \sqcup \mathbb{G}_i^n) \equiv_{\mathrm{m}} \mathbb{H}_2^n \sqcup (\mathbb{R}^n \sqcup \mathbb{G}_i^n)$ and

(iv) $(\mathbb{H}_2^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_i^n \equiv_{\mathrm{m}} \mathbb{HR}^n \sqcup \mathbb{G}_i^n$.

As $\mathbb{G}_1^n$ generates all directed graphs, verifying (iii) (respectively (iv)) for $i = 1$ actually verifies $\mathbb{H}_1^n \sqcup \mathbb{R}^n \equiv_{\mathrm{m}} \mathbb{H}_2^n \sqcup \mathbb{R}^n$ (respectively $\mathbb{H}_2^n \sqcup \mathbb{R}^n \equiv_{\mathrm{m}} \mathbb{HR}^n$) and modular equivalence for the other generator modules follows from Corollary 5.19. The motivation, however, is to see whether it is faster to verify a weaker result, that is, to verify the equivalence in the context of certain subclasses of directed graphs.

The average running times for this experiment are presented in Figures 7.8 and 7.9. Both equivalences are harder to verify in the contexts of symmetric and irreflexive graphs, and graphs with Euclidean edge relation than in other contexts. Using symmetric and irreflexive graphs as context turned out to be especially time consuming when verifying equivalence in (iv). We used a timeout of 6000 seconds in this experiment and were not able to verify (iv) for the values $n = 9$ and $n = 10$ with this limit. For the sake of clarity, we also dropped the average time needed to verify $(\mathbb{H}_2^8 \sqcup \mathbb{R}^8) \sqcup \mathbb{G}_3^8 \equiv_{\mathrm{m}} \mathbb{HR}^8 \sqcup \mathbb{G}_3^8$ (4048 seconds) from Figure 7.9. The average numbers of choice points behave similarly to the average running times. The somewhat surprising impli-

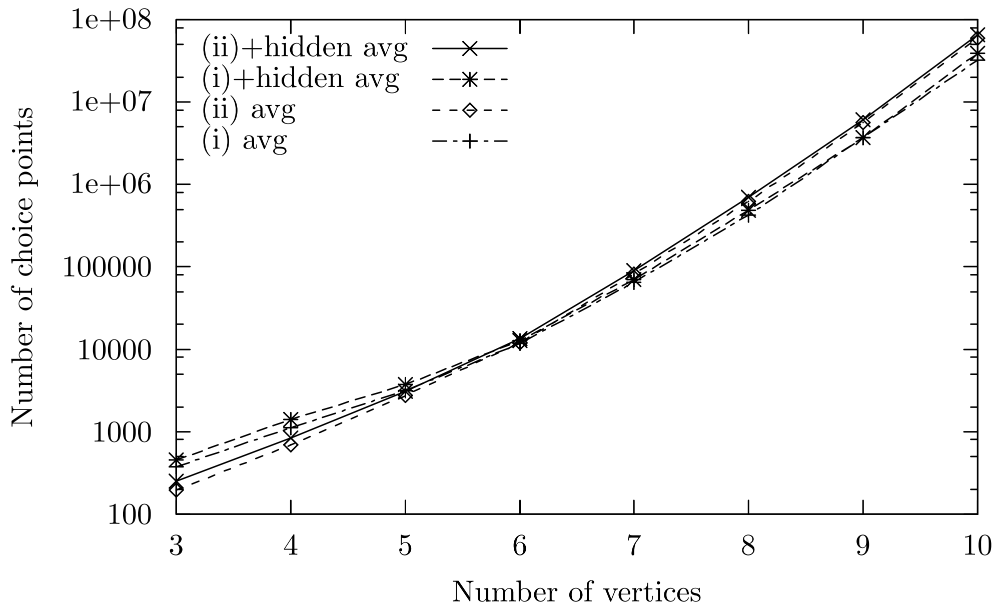

Figure 7.7: The average numbers of choice points for verifying (i) $\mathbb{H}_1^n \sqcup (\mathbb{R}^n \sqcup \mathbb{G}_1^n) \equiv_{\mathrm{m}} \mathbb{H}_2^n \sqcup (\mathbb{R}^n \sqcup \mathbb{G}_1^n)$ plus $(\mathbb{H}_2^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_1^n \equiv_{\mathrm{m}} \mathbb{HR}^n \sqcup \mathbb{G}_1^n$, and (ii) $(\mathbb{H}_1^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_1^n \equiv_{\mathrm{m}} \mathbb{HR}^n \sqcup \mathbb{G}_1^n$ with predicate `reached(X)` hidden/visible in the second Hamiltonian cycle experiment.

cation of this experiment is that restricting the number of possible inputs by applying a graph generator having less stable models than $\mathbb{G}_1^n$ does not necessarily make the equivalence verification task more efficient. The reason for this might be that it can be more difficult to find stable models for a more specific generator module.

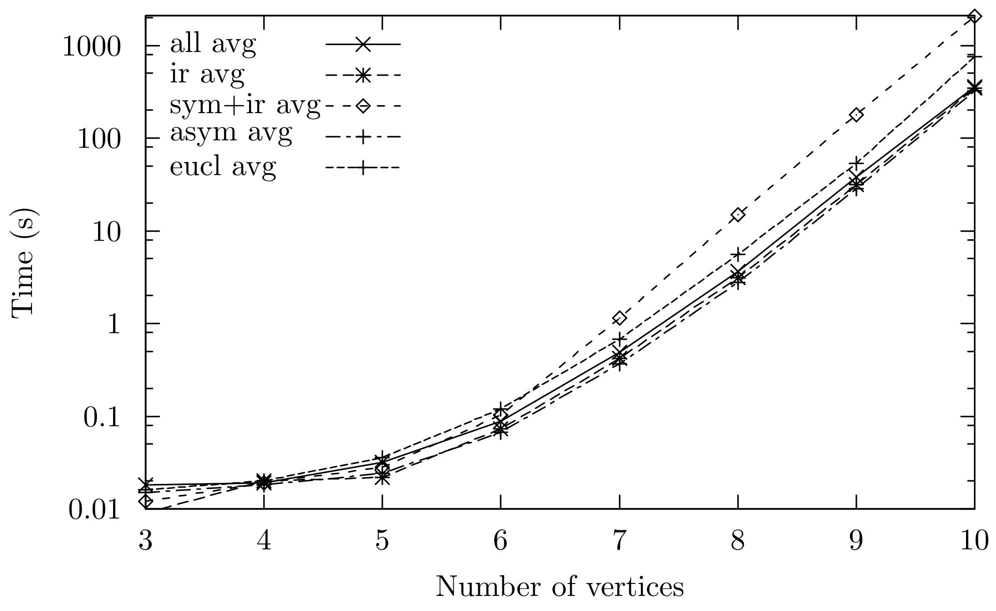

Figure 7.8: The average running times for verifying (iii) $\mathbb{H}_1^n \sqcup (\mathbb{R}^n \sqcup \mathbb{G}_i^n) \equiv_{\mathrm{m}} \mathbb{H}_2^n \sqcup (\mathbb{R}^n \sqcup \mathbb{G}_i^n)$ for $i = 1$ (all graphs), 2 (irreflexive graphs), 3 (symmetric and irreflexive graphs), 4 (asymmetric graphs), 5 (graphs with Euclidean edge relation) in the third Hamiltonian cycle experiment.

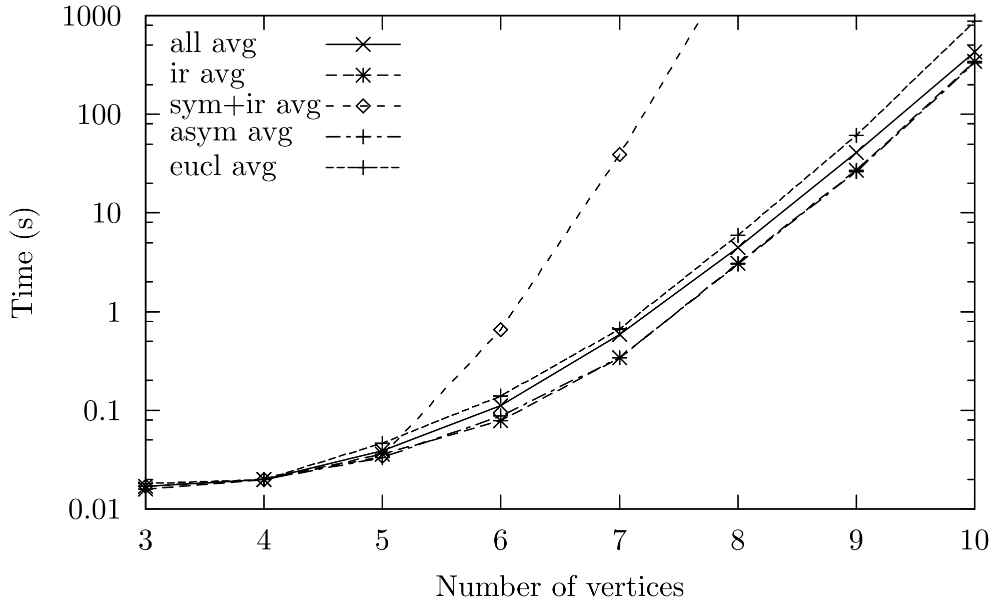

Figure 7.9: The average running times for verifying (iv) $(\mathbb{H}_2^n \sqcup \mathbb{R}^n) \sqcup \mathbb{G}_i^n \equiv_{\mathrm{m}} \mathbb{H}\mathbb{R}^n \sqcup \mathbb{G}_i^n$ for $i = 1$ (all graphs), 2 (irreflexive graphs), 3 (symmetric and irreflexive graphs), 4 (asymmetric graphs), 5 (graphs with Euclidean edge relation) in the third Hamiltonian cycle experiment.

# 8 CONCLUSIONS

In this report we establish a simple and intuitive notion for a logic program module that interacts through an input/output interface. This is achieved by accommodating program modules proposed by Gaifman and Shapiro [17] to the context of answer set programming. Full compatibility of the module system and the stable model semantics is achieved by allowing positive recursion to occur inside modules only.

One of the main results is the *module theorem* in Section 3.3 showing that module-level stability implies program-level stability, and vice versa, as long as the stable models of the submodules are compatible. Even though the module theorem is a theoretical result motivating the feasibility of our module system, it is also a tool for structuring and simplifying mathematical arguments used in many of the proofs in Appendix A. The module theorem also generalizes the splitting set theorem [39] in the case of normal logic programs and SMODELS programs as our result allows negative recursion between modules.

We introduce a notion of modular equivalence that is a proper congruence relation for composition of modules, and show that deciding modular equivalence is **coNP**-complete for modules with the EVA property, that is, for modules that have enough visible atoms so that one can distinguish their stable models based on the visible part.

We extend the translation-based method for verification of visible equivalence proposed in [31] to cover the translation-based verification of modular equivalence. In this way, the verification of modular equivalence can be accomplished with existing methods and specialized solvers need not be developed. We also propose strategies to exploit modular verification techniques, and consider questions involved in the problem of finding a suitable module structure for a program when there is no explicit a priori knowledge on the underlying structure.

Finally, we evaluate experimentally the efficiency of the translation-based method in the verification of modular equivalence. Based on the experimental evaluation, modularization of the verification of (modular) equivalence seems to be a good idea in many cases, especially if the common context shared by the modules is large and the number of submodules stays reasonable.

## 8.1 FUTURE WORK

Several further questions still remain to be considered, for instance, how to characterize semantically the syntactical restrictions for module composition, and is it possible to weaken the conditions insisted upon module composition, for example, by allowing positive recursion between modules under some conditions. It is still unclear whether modular equivalence has a model theoretic characterization similar to SE-models [69] and UE-models [7] for strong and uniform equivalence, respectively.

The current results should be extended to cover even more general classes of logic programs, such as disjunctive logic programs [22, 62], nested logic programs [38] and weight constraint programs [65]. A way to achieve this is to find a modularly equivalent translation to SMODELS or normal logic program modules similarly to the strategy used in Chapter 5, where we extend our theoretical results for normal logic program modules to cover the class of SMODELS program modules.

Current ASP solvers have no support for modularity, and as seen in Chapter 7, workarounds are needed to simulate modular ASP with LPARSE and SMODELS. To promote modular answer set programming it is necessary to be able to do modular ASP in practice. Incorporating modularity into current solvers is thus crucial. Certain aspects are easy to implement, for example, the current input language of LPARSE has already declarations for defining the hidden and visible Herbrand bases of a program. To support modular ASP, one should further partition the visible Herbrand base into input and output. There are also more involved questions. Note that the input atoms are not defined inside a module and thus one should be able to tell from which module the definitions for the input atoms are imported. A metalanguage for defining the modular aspects and whether modules are seen as libraries and/or macros should be agreed upon. It needs to be considered how to do grounding for a stand-alone module. To ease the task of handling several modules it would be desirable to have several modules in one file.

Programs that solve real-life problems can be very large and complex, and it becomes necessary to be able to structure the programs into modules. We see that it is not very likely that any single constraint programming approach proves out to be the most efficient for solving all kinds of hard combinatorial problems. Therefore there is a need for a well-defined *module interface* that gives support for other constraint programming approaches, such as, for example, *propositional satisfiability* (SAT) and *constraint programming* (CSP), in addition to ASP. This general *module-oriented constraint programming paradigm* would then enable a strategy where different constraint programming approaches and solvers are used, possibly *in parallel*, to solve the subproblems utilizing the characteristics and best features of each approach. The solution to the problem would then be composed from the solutions for the individual subproblems.

The module system proposed in this report is a step in the direction of modular answer set programming. So far we have limited ourselves to *propositional* logic program modules under the stable model semantics. In practice it might, however, be impossible or undesirable to consider only ground instances of modules. A crucial step further is to extend the concept of modularity to the *non-ground* case, that is, to consider program modules involving *variables*. To extend the module system to the non-ground case is a nontrivial task, and several aspects need to be considered, for example, how to define the input/output interface and interaction of modules (using predicates, grounded atoms or constants, etc.), how to define the concept of *modular equivalence* for non-ground modules and still maintain the essential *congruence property* with respect to *composition* of modules.

An important aspect to consider is *what are good programming practices* for module-oriented constraint programming. For example, in the case of propositional modular ASP, handling positive recursion inside modules, and considering modules that have enough visible atoms can be seen as good ways to structure programs.

## ACKNOWLEDGEMENTS

This is a reprint of my Licentiate's thesis [55]. The thesis is result of studies and research in Laboratory for Theoretical Computer Science at Helsinki University of Technology. I would like to thank my instructor Docent Tomi Janhunen for his advice and comments during the process of writing this thesis. I am grateful to the head of the laboratory and my supervisor Professor Ilkka Niemel  for his comments and for giving me the opportunity to concentrate to this work. I also thank all my colleagues in the laboratory for creating an inspiring working atmosphere.

Research reported in this thesis is partially funded by the Academy of Finland (project #211025). The financial support from Helsinki Graduate School in Computer Science and Engineering, Nokia Foundation, Finnish Foundation of Technology (TES) and Kainuu Foundation of the Finnish Cultural Foundation is gratefully acknowledged.

# A PROOFS

## A.1 PROOFS FOR CHAPTER 3

In this section, the proofs involve *normal logic programs* and *normal logic program modules* unless otherwise stated.

**Proof of Theorem 3.9** To prove Theorem 3.9 we apply the following Lemmas A.1 and A.2, and note that $M \cap \mathrm{Hb}(\mathbb{P}_1)$ and $M \cap \mathrm{Hb}(\mathbb{P}_2)$ in Lemma A.1 are compatible. $\square$

**Lemma A.1** *Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be modules such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined. If $M \in \mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$, then $M \cap \mathrm{Hb}(\mathbb{P}_1) \in \mathrm{SM}(\mathbb{P}_1)$ and $M \cap \mathrm{Hb}(\mathbb{P}_2) \in \mathrm{SM}(\mathbb{P}_2)$.*

**Proof of Lemma A.1** Let $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$ be modules such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined. Denote

$$\mathbb{P} = \mathbb{P}_1 \sqcup \mathbb{P}_2 = (P_1 \cup P_2, (I_1 \setminus O_2) \cup (I_2 \setminus O_1), O_1 \cup O_2) = (P, I, O).$$

Consider arbitrary $M \in \mathrm{SM}(\mathbb{P})$, that is, $M = \mathrm{LM}(P^M \cup \mathrm{F}_{M \cap I})$. Define $M_1 = M \cap \mathrm{Hb}(\mathbb{P}_1)$ and $M_2 = M \cap \mathrm{Hb}(\mathbb{P}_2)$. It holds that $M_1 \cup M_2 = M$. Note that, since the intersection of Herbrand bases of $\mathbb{P}_1$ and $\mathbb{P}_2$ (see equation (3.1)) is not necessarily empty, also $M_1 \cap M_2 = \emptyset$ does not necessarily hold. Since $M = M_1 \cup M_2$ is the least model of $P^M \cup \mathrm{F}_{M \cap I}$,

$$M_1 \cup M_2 \models (P_1 \cup P_2)^{M_1 \cup M_2} \cup \mathrm{F}_{(M_1 \cup M_2) \cap I}. \tag{A.1}$$

By restricting (A.1) to $\mathrm{Hb}(\mathbb{P}_i)$, we get $M_i \models {P_i}^{M_i} \cup \mathrm{F}_{M_i \cap I}$ for $i \in \{1, 2\}$, and furthermore $M_i \models {P_i}^{M_i}$. Since $M_i \cap I_i \subseteq M_i$, we can conclude

$$M_i \models {P_i}^{M_i} \cup \mathrm{F}_{M_i \cap I_i} \text{ for } i \in \{1, 2\}. \tag{A.2}$$

Assume that $M_1 \neq \mathrm{LM}({P_1}^{M_1} \cup \mathrm{F}_{M_1 \cap I_1})$, that is, there is $N_1 \subset M_1$ such that $N_1 \models {P_1}^{M_1} \cup \mathrm{F}_{M_1 \cap I_1}$. Define $A = M_1 \setminus N_1 \neq \emptyset$. Let us show that

$$N_1 \cup (M_2 \setminus A) \models P^M \cup \mathrm{F}_{M \cap I} = ({P_1}^{M_1} \cup \mathrm{F}_{M_1 \cap I}) \cup ({P_2}^{M_2} \cup \mathrm{F}_{M_2 \cap I}). \tag{A.3}$$

First, note that $N_1$ and $M_2 \setminus A$ are compatible.

- By assumption $N_1 \models {P_1}^{M_1} \cup \mathrm{F}_{M_1 \cap I_1}$. Since

  $$M_1 \cap I = M_1 \cap ((I_1 \setminus O_2) \cup (I_2 \setminus O_1)) \subseteq M_1 \cap I_1 \subseteq N_1,$$

  it holds $N_1 \models {P_1}^{M_1} \cup \mathrm{F}_{M_1 \cap I}$.

- Recall equation (3.1). Since $M_1 \cap I_1 \subseteq N_1$ and $A \subseteq \mathrm{Hb}(\mathbb{P}_1)$, it holds $A \cap \mathrm{Hb}(\mathbb{P}_2) \subseteq O_1 \cap I_2$. Since the atoms in $I_2$ can appear only in positive bodies of the rules in ${P_2}^{M_2}$ and $M_2 \models {P_2}^{M_2}$, it holds that $M_2 \setminus A \models {P_2}^{M_2}$. Furthermore, since $A \cap I = \emptyset$, we get $M_2 \setminus A \models {P_2}^{M_2} \cup \mathrm{F}_{M_2 \cap I}$.

Now, $N_1 \cup (M_2 \setminus A) \subset M$ and (A.3) holds, which is contradictory to $M = \mathrm{LM}(P^M \cup \mathrm{F}_{M \cap I})$. Thus $M_1 \in \mathrm{SM}(\mathbb{P}_1)$. It follows by symmetry that $M_2 \in \mathrm{SM}(\mathbb{P}_2)$. $\square$

**Lemma A.2** *Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be modules such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined. If $M_1 \in \mathrm{SM}(\mathbb{P}_1)$ and $M_2 \in \mathrm{SM}(\mathbb{P}_2)$ are compatible, then $M_1 \cup M_2 \in \mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$.*

**Proof of Lemma A.2** Consider modules $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$ such that their join

$$\mathbb{P} = \mathbb{P}_1 \sqcup \mathbb{P}_2 = (P_1 \cup P_2, (I_1 \setminus O_2) \cup (I_2 \setminus O_1), O_1 \cup O_2) = (P, I, O)$$

is defined. Denote $H_1 = \mathrm{Hb_h}(\mathbb{P}_1)$ and $H_2 = \mathrm{Hb_h}(\mathbb{P}_2)$. Assume $M_1 \in \mathrm{SM}(\mathbb{P}_1)$, $M_2 \in \mathrm{SM}(\mathbb{P}_2)$ and $M_1 \cap \mathrm{Hb_v}(\mathbb{P}_2) = M_2 \cap \mathrm{Hb_v}(\mathbb{P}_1)$, that is, $M_1$ and $M_2$ are compatible. Thus it holds

$$\begin{aligned} M_1 \cap (I_1 \cap I_2) &= M_2 \cap (I_1 \cap I_2), \\ M_1 \cap (I_1 \cap O_2) &= M_2 \cap (I_1 \cap O_2), \text{ and} \\ M_1 \cap (O_1 \cap I_2) &= M_2 \cap (O_1 \cap I_2). \end{aligned}$$

Define $M = M_1 \cup M_2$. Since $M_1$ and $M_2$ are compatible, $M_1 = M \cap \mathrm{Hb}(\mathbb{P}_1)$, $M_2 = M \cap \mathrm{Hb}(\mathbb{P}_2)$ and $P^M = {P_1}^{M_1} \cup {P_2}^{M_2}$. Now $M_1 = \mathrm{LM}({P_1}^{M_1} \cup \mathrm{F}_{M_1 \cap I_1})$, $M_2 = \mathrm{LM}({P_2}^{M_2} \cup \mathrm{F}_{M_2 \cap I_2})$ and we want to show that $M = \mathrm{LM}(P^M \cup \mathrm{F}_{M \cap I})$.

Let $C_1 < C_2 < \cdots < C_n$ be a strict total order for the strongly connected components of $\mathrm{Dep}^+(P)$ (such an order always exists). First, we show that exactly one of the following holds for each $C_i$ $(1 \leq i \leq n)$:

(1a) $C_i \cap I \neq \emptyset$, or

(1b) $C_i \cap (O_1 \cup H_1) \neq \emptyset$, or

(1c) $C_i \cap (O_2 \cup H_2) \neq \emptyset$.

As $\emptyset \subset C_i \subseteq \mathrm{Hb}(\mathbb{P})$ and $\mathrm{Hb}(\mathbb{P}) = I \cup (O_1 \cup H_1) \cup (O_2 \cup H_2)$, at least one of the conditions must hold.

Assume $C_i \cap I \neq \emptyset$ and $C_i \cap (O_1 \cup H_1) \neq \emptyset$. The atoms in $I$ do not appear in the heads of the rules in $P$ and $I \cap (O_1 \cup H_1) = \emptyset$. Thus it is impossible for $C_i$ to be a strongly connected component of $\mathrm{Dep}^+(P)$. This is contradictory to the assumption. Similarly, assuming $C_i \cap I \neq \emptyset$ and $C_i \cap (O_2 \cup H_2) \neq \emptyset$ leads to contradiction.

Assume that $C_i \cap (O_1 \cup H_1) \neq \emptyset$ and $C_i \cap (O_2 \cup H_2) \neq \emptyset$. Since $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined, there is no positive recursion between the modules, and by Definition 3.2 either $O_1 \cap C_i = \emptyset$ or $O_2 \cap C_i = \emptyset$ (or both) holds. Assume without loss of generality $O_2 \cap C_i = \emptyset$ which implies $H_2 \cap C_i \neq \emptyset$. The atoms in $H_2$ do not appear in the rules of $P_1$ and since $O_2 \cap C_i = \emptyset$ there can be no dependence between $H_2$ and $O_1 \cup H_1$. Thus $C_i$ is not a strongly connected component, which is again contradictory to the assumption.

It is easy to show that exactly one of conditions (1a), (1b) and (1c) holds for $C_i$ if and only if exactly one of the following holds:

(2a) $C_i \subseteq I$, or

(2b) $C_i \subseteq O_1 \cup H_1$, or

(2c) $C_i \subseteq O_2 \cup H_2$.

Notice that if condition (2a) holds, then $C_i$ is a singleton set.

Next, we consider $(P^M \cup \mathrm{F}_{M\cap I})[C_i]$, that is, the set of rules in $P^M \cup \mathrm{F}_{M\cap I}$ defining the atoms in $C_i$. As exactly one of the conditions (2a), (2b) and (2c) holds, exactly one of the following applies:

(3a) If $C_i \subseteq I$, then $(P^M \cup \mathrm{F}_{M\cap I})[C_i] = \mathrm{F}_{M\cap I\cap C_i} = \mathrm{F}_{M\cap C_i}$.

(3b) If $C_i \subseteq O_1 \cup H_1$, then
$$(P^M \cup \mathrm{F}_{M\cap I})[C_i] = P^M[C_i] = {P_1}^{M_1}[C_i] \cup \underbrace{{P_2}^{M_2}[C_i]}_{=\emptyset} = {P_1}^{M_1}[C_i].$$

(3c) If $C_i \subseteq O_2 \cup H_2$, then
$$(P^M \cup \mathrm{F}_{M\cap I})[C_i] = P^M[C_i] = \underbrace{{P_1}^{M_1}[C_i]}_{=\emptyset} \cup {P_2}^{M_2}[C_i] = {P_2}^{M_2}[C_i].$$

Finally, we show by induction that
$$M \cap (\bigcup_{i=1}^{k} C_i) = \mathrm{LM}((P^M \cup \mathrm{F}_{M\cap I})[\bigcup_{i=1}^{k} C_i]) \tag{A.4}$$
for all $0 \le k \le n$.

**Basis.** $M \cap \emptyset = \mathrm{LM}(\emptyset)$.

**Inductive hypothesis.** Assume $M \cap (\bigcup_{i=1}^{k-1} C_i) = \mathrm{LM}((P^M \cup \mathrm{F}_{M\cap I})[\bigcup_{i=1}^{k-1} C_i])$.

**Inductive step.** There are three possible cases, corresponding to (2a), (2b) and (2c). In these cases (3a), (3b) and (3c), respectively, are applied. In the following, Theorem 2.7 (splitting set theorem) is applied with respect to a splitting set $U_k = \bigcup_{i=1}^{k-1} C_i$.

(i) If $C_k \subseteq I$, then
$$\begin{aligned}
&\mathrm{LM}((P^M \cup \mathrm{F}_{M\cap I})[\bigcup_{i=1}^{k} C_i]) \\
\overset{(3a)}{=}\ & \mathrm{LM}((P^M \cup \mathrm{F}_{M\cap I})[\bigcup_{i=1}^{k-1} C_i] \cup (\mathrm{F}_{M\cap I} \cap C_k)) \\
\overset{\text{T. 2.7}}{=}\ & \mathrm{LM}((P^M \cup \mathrm{F}_{M\cap I})[\bigcup_{i=1}^{k-1} C_i]) \cup (M \cap C_k) \\
\overset{\text{I.H.}}{=}\ & (M \cap (\bigcup_{i=1}^{k-1} C_i)) \cup (M \cap C_k) \\
=\ & M \cap (\bigcup_{i=1}^{k} C_i).
\end{aligned}$$

(ii) If $C_k \subseteq O_1 \cup H_1$, then
$$\begin{aligned}
&\mathrm{LM}((P^M \cup \mathrm{F}_{M\cap I})[\bigcup_{i=1}^{k} C_i]) \\
\overset{(3b)}{=}\ & \mathrm{LM}((P^M \cup \mathrm{F}_{M\cap I})[\bigcup_{i=1}^{k-1} C_i] \cup {P_1}^{M_1}[C_k]) \\
\overset{\text{T. 2.7}}{=}\ & \mathrm{LM}((P^M \cup \mathrm{F}_{M\cap I})[\bigcup_{i=1}^{k-1} C_i]) \\
& \qquad \cup \mathrm{LM}(\mathrm{e}({P_1}^{M_1}[C_k], \mathrm{LM}((P^M \cup \mathrm{F}_{M\cap I})[\bigcup_{i=1}^{k-1} C_i]))) \\
\overset{\text{I.H.}}{=}\ & M \cap (\bigcup_{i=1}^{k-1} C_i) \cup \mathrm{LM}(\mathrm{e}({P_1}^{M_1}[C_k], M \cap (\bigcup_{i=1}^{k-1} C_i))) \\
=\ & M \cap (\bigcup_{i=1}^{k-1} C_i) \cup \mathrm{LM}(\mathrm{e}({P_1}^{M_1}[C_k], M_1 \cap (\bigcup_{i=1}^{k-1} C_i))) \\
=\ & M \cap (\bigcup_{i=1}^{k-1} C_i) \cup (M_1 \cap C_k) \ = \ M \cap (\bigcup_{i=1}^{k} C_i).
\end{aligned}$$

(iii) The case $C_k \subseteq O_2 \cup H_2$ is symmetrical to (ii).

Thus equation (A.4) holds for all $0 \leq k \leq n$. For $k = n$ we get $M = \mathrm{LM}(P^M \cup \mathrm{F}_{M \cap I})$. $\square$

## A.2 PROOFS FOR CHAPTER 4

In this section, the proofs involve *normal logic programs* and *normal logic program modules* unless otherwise stated.

**Proof of Theorem 4.2** Let $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$ be modules such that $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$. Let $\mathbb{R} = (R, I_R, O_R)$ be an arbitrary module such that $\mathbb{P} \sqcup \mathbb{R}$ and $\mathbb{Q} \sqcup \mathbb{R}$ are defined. Consider an arbitrary $M \in \mathrm{SM}(\mathbb{P} \sqcup \mathbb{R})$. By Theorem 3.9 $M_P = M \cap \mathrm{Hb}(\mathbb{P}) \in \mathrm{SM}(\mathbb{P})$ and $M_R = M \cap \mathrm{Hb}(\mathbb{R}) \in \mathrm{SM}(\mathbb{R})$. Since $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$, there is a bijection $f : \mathrm{SM}(\mathbb{P}) \to \mathrm{SM}(\mathbb{Q})$ such that $M_P \in \mathrm{SM}(\mathbb{P}) \iff f(M_P) \in \mathrm{SM}(\mathbb{Q})$, and

$$M_P \cap (O \cup I) = f(M_P) \cap (O \cup I). \tag{A.5}$$

Denote $M_Q = f(M_P)$. Clearly, $M_P$ and $M_R$ are compatible. Since (A.5) holds, also $M_Q$ and $M_R$ are compatible. Applying Theorem 3.9 we get $M_Q \cup M_R \in \mathrm{SM}(\mathbb{Q} \sqcup \mathbb{R})$.

Now, define function $g : \mathrm{SM}(\mathbb{P} \sqcup \mathbb{R}) \to \mathrm{SM}(\mathbb{Q} \sqcup \mathbb{R})$ as

$$g(M) = f(M \cap \mathrm{Hb}(\mathbb{P})) \cup (M \cap \mathrm{Hb}(\mathbb{R})).$$

Clearly, $g$ restricted to the visible part is an identity function, that is,

$$M \cap (I \cup I_R \cup O \cup O_R) = g(M) \cap (I \cup I_R \cup O \cup O_R).$$

Function $g$ is a bijection, since

- $g$ is an injection: $M \neq N$ implies $g(M) \neq g(N)$ for all $M, N \in \mathrm{SM}(\mathbb{P} \sqcup \mathbb{R})$, since $f(M \cap \mathrm{Hb}(\mathbb{P})) \neq f(N \cap \mathrm{Hb}(\mathbb{P}))$ or $M \cap \mathrm{Hb}(\mathbb{R}) \neq N \cap \mathrm{Hb}(\mathbb{R})$.

- $g$ is a surjection: for any $M \in \mathrm{SM}(\mathbb{Q} \sqcup \mathbb{R})$, $N = f^{-1}(M \cap \mathrm{Hb}(\mathbb{Q})) \cup (M \cap \mathrm{Hb}(\mathbb{R})) \in \mathrm{SM}(\mathbb{P} \sqcup \mathbb{R})$ and $g(N) = M$, since $f$ is a surjection.

The inverse function $g^{-1} : \mathrm{SM}(\mathbb{Q} \sqcup \mathbb{R}) \to \mathrm{SM}(\mathbb{P} \sqcup \mathbb{R})$ can be defined as $g^{-1}(N) = f^{-1}(N \cap \mathrm{Hb}(\mathbb{Q})) \cup (N \cap \mathrm{Hb}(\mathbb{R}))$. Thus $\mathbb{P} \sqcup \mathbb{R} \equiv_{\mathrm{m}} \mathbb{Q} \sqcup \mathbb{R}$. $\square$

## A.3 PROOFS FOR CHAPTER 5

In this section, the proofs involve SMODELS *programs* and SMODELS *program modules* unless otherwise stated. Since basic rules and constraint rules are special cases of weight rules, we will assume that SMODELS programs and SMODELS program modules consist of choice rules, weight rules and compute statements only.

**Proof of Theorem 5.15** Consider arbitrary SMODELS program modules $\mathbb{P}_1 = (P_1, I_1, O_1)$ and $\mathbb{P}_2 = (P_2, I_2, O_2)$ such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined, that is, $O_1 \cap O_2 = \emptyset$, $\mathrm{Hb_h}(\mathbb{P}_1)$ is local to $\mathbb{P}_1$, $\mathrm{Hb_h}(\mathbb{P}_2)$ is local to $\mathbb{P}_2$, and there is no positive recursion between $\mathbb{P}_1$ and $\mathbb{P}_2$.

Clearly, $O_1 \cap O_2$ holds for translations $\mathrm{Tr_{NLP}}(\mathbb{P}_1) = (\mathrm{Tr_{NLP}}(P_1), I_1, O_1)$ and $\mathrm{Tr_{NLP}}(\mathbb{P}_2) = (\mathrm{Tr_{NLP}}(P_2), I_2, O_2)$. Translation $\mathrm{Tr_{NLP}}$ produces a new (hidden) atom $\overline{a}$ for each atom $a$ appearing in the head of a choice rule and $f_{P_i}$ $(i = 1, 2)$ for translating compute statements. Since these are new atoms, it is reasonable to assume that hidden atoms in $\mathrm{Tr_{NLP}}(\mathbb{P}_1)$ and $\mathrm{Tr_{NLP}}(\mathbb{P}_2)$ remain local, too.

Weight and choice rules generate the edges in $\mathrm{Dep}^+(P_1 \cup P_2)$. With respect to the choice rules in $P_1 \cup P_2$ the dependency graph $\mathrm{Dep}^+(\mathrm{Tr_{NLP}}(P_1) \cup \mathrm{Tr_{NLP}}(P_2))$ contains the same edges as $\mathrm{Dep}^+(P_1 \cup P_2)$. An additional vertex is added for each new atom $\overline{a}$, but no edges are added because all the new dependencies introduced in $\mathrm{Tr_{NLP}}$ are negative. With respect to the weight rules, $\langle a, b\rangle \in \mathrm{Dep}^+(P_1 \cup P_2)$ if and only if $\langle a, b\rangle \in \mathrm{Dep}^+(\mathrm{Tr_{NLP}}(P_1) \cup \mathrm{Tr_{NLP}}(P_2))$. The rules in $\mathrm{Tr_{NLP}}(P_1) \cup \mathrm{Tr_{NLP}}(P_2)$ corresponding to compute statements do not yield positive recursion either. Thus there is no positive recursion between $\mathrm{Tr_{NLP}}(\mathbb{P}_1)$ and $\mathrm{Tr_{NLP}}(\mathbb{P}_2)$, and $\mathrm{Tr_{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr_{NLP}}(\mathbb{P}_2)$ is defined.

It is straightforward to see that $\mathrm{Tr_{NLP}}(P_1) \cup \mathrm{Tr_{NLP}}(P_2)$ and $\mathrm{Tr_{NLP}}(P_1 \cup P_2)$ contain exactly the same rules with respect to the choice and the weight rules in $P_1 \cup P_2$. Recall that new atoms are introduced for atoms appearing in the heads of choice rules. Because $(O_1 \cup \mathrm{Hb_h}(\mathbb{P}_1)) \cap (O_2 \cup \mathrm{Hb_h}(\mathbb{P}_2)) = \emptyset$, there can be no conflict. The translation of compute statements introduces new atoms $f_{P_1}$ and $f_{P_2}$ in $\mathrm{Tr_{NLP}}(P_1) \cup \mathrm{Tr_{NLP}}(P_2)$ and a new atom $f_{P_1 \cup P_2}$ in $\mathrm{Tr_{NLP}}(P_1 \cup P_2)$. However, it is easy to see that $\mathrm{SM}(\mathrm{Tr_{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr_{NLP}}(\mathbb{P}_2)) = \mathrm{SM}(\mathrm{Tr_{NLP}}(\mathbb{P}_1 \sqcup \mathbb{P}_2))$ as atoms $f_{P_1}$, $f_{P_2}$ and $f_{P_1 \cup P_2}$ are always false in a stable model.

Finally note that translation $\mathrm{Tr_{NLP}}$ does not affect the input and output sets. Thus, they are the same for $\mathrm{Tr_{NLP}}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$ and $\mathrm{Tr_{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr_{NLP}}(\mathbb{P}_2)$, and we have $\mathrm{Tr_{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr_{NLP}}(\mathbb{P}_2) \equiv_{\mathrm{m}} \mathrm{Tr_{NLP}}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$. □

**Proof of Theorem 5.16** Let $\mathbb{P} = (P, I, O)$ be an SMODELS program module and $\mathrm{Tr_{NLP}}(\mathbb{P}) = (\mathrm{Tr_{NLP}}(P), I, O)$ defined as in Definition 5.13. Consider an arbitrary $M \in \mathrm{SM}(\mathbb{P})$, that is, $M = \mathrm{LM}(P^M \cup \mathrm{F}_{M\cap I})$ and $M \models \mathrm{CompS}(P)$. Define

$$N = M \cup \{\overline{a} \mid \{H\} \leftarrow B^+, {\sim}B^- \in P \text{ and } a \in H \setminus M\}.$$

First note that $\overline{a} \in N$ if and only if $a \notin M$ if and only if $a \notin N$ for atoms $a$ appearing in the head of a choice rule in $P$. We consider the satisfaction of the rules in $\mathrm{Tr}_{\mathrm{NLP}}(P)^N$. Consider an arbitrary rule $r_1 = h \leftarrow C \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$ such that $h \in \mathrm{Hb}(\mathbb{P})$. There are two possibilities:

- There is a rule $h \leftarrow C, \sim D, \sim\overline{h} \in \mathrm{Tr}_{\mathrm{NLP}}(P)$ corresponding to a choice rule $\{H\} \leftarrow C, \sim D \in P$ such that $h \in H$, $\overline{h} \notin N$ (which implies $h \in N$), and $D \cap N = \emptyset$. Since $h \in N$, it holds that $N \models r_1$.

- There is a rule $h \leftarrow C, \sim D \in \mathrm{Tr}_{\mathrm{NLP}}(P)$ corresponding to a weight rule $r_2 = h \leftarrow w \leq \{B^+ = W_{B^+}, \sim B^- = W_{B^-}\} \in P$ such that $C \subseteq B^+$, $D \subseteq B^-$, $w \leq \sum_{c \in C} w_c + \sum_{d \in D} w_d$ and $D \cap N = \emptyset$. If $h \in N$, then $N \models r_1$. If $h \notin N$, assume that $N \not\models r_1$, that is, $C \subseteq N$. This implies $h \notin M, C \subseteq M$ and $D \cap M = \emptyset$ and $w \leq \mathrm{WS}_M(B^+ = W_{B^+}, \sim B^- = W_{B^-})$. Thus $M \not\models r_2$ which is in contradiction with $M \models P$ (notice that if $M \models P^M$ and $M \models \mathrm{CompS}(P)$, then $M \models P$). Therefore $N \models r_1$.

Rest of the rules in $\mathrm{Tr}_{\mathrm{NLP}}(P)^N$ are one of the following forms.

- A fact $\overline{h} \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$ if there is a rule $\overline{h} \leftarrow \sim h \in \mathrm{Tr}_{\mathrm{NLP}}(P)$, that is, $h$ appears in the head of a choice rule in $P$, and $h \notin N$. Since $h \notin N$ implies $\overline{h} \in N$, then $N \models \overline{h}$.

- A rule $r = f_P \leftarrow b \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$ if $f_P \notin N$ and there is a compute statement compute $\{B^+, \sim B^-\} \in P$ such that $b \in B^-$. Since $M \models \mathrm{CompS}(P)$ we have $B^- \cap M = B^- \cap N = \emptyset$ and $N \models r$.

- A fact $f_P \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$ if $f_P \notin N$ and there is a compute statement compute $\{B^+, \sim B^-\} \in P$ such that $a \in B^+$ and $a \notin N$. Since $M \models \mathrm{CompS}(P)$ we have $B^+ \subseteq M \subseteq N$ and there cannot be a fact $f_P$ in $\mathrm{Tr}_{\mathrm{NLP}}(P)^N$.

Thus $N \models \mathrm{Tr}_{\mathrm{NLP}}(P)^N \cup \mathrm{F}_{N \cap I}$. Assume now $N \neq \mathrm{LM}(\mathrm{Tr}_{\mathrm{NLP}}(P)^N \cup \mathrm{F}_{N \cap I})$, that is, there is $N' \subset N$ such that $N' \models \mathrm{Tr}_{\mathrm{NLP}}(P)^N \cup \mathrm{F}_{N \cap I}$. Clearly $N' \cap I = N \cap I = M \cap I$. We define $M' = N' \cap \mathrm{Hb}(\mathbb{P})$ and show $M' \models P^M$.

- If $r_1 = h \leftarrow B^+ \in P^M$ then there is a choice rule $\{H\} \leftarrow B^+, \sim B^- \in P$, such that $B^- \cap M = \emptyset$ and $h \in M \cap H$. Since $B^- \cap M = \emptyset$ implies $B^- \cap N = \emptyset$, and $h \in M$ implies $h \in N$ and $\overline{h} \notin N$, there is a rule $h \leftarrow B^+ \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$. Since $N' \models h \leftarrow B^+$ and $N' \cap \mathrm{Hb}(\mathbb{P}) = M' \cap \mathrm{Hb}(\mathbb{P})$, it holds $M' \models r_1$.

- If $r_2 = h \leftarrow w' \leq \{B^+ = W_{B^+}\} \in P^M$ then there is a weight rule $h \leftarrow w \leq \{B^+ = W_{B^+}, \sim B^- = W_{B^-}\} \in P$ such that $w' = \max(0, w - \mathrm{WS}_M(\sim B^- = W_{B^-}))$. Assume $M' \not\models r_2$, that is, $h \notin M'$ and $w' \leq \mathrm{WS}_{M'}(B^+ = W_{B^+})$. Define $D = B^- \setminus M$ and $C = B^+ \cap M'$, and recall that $N' \cap \mathrm{Hb}(\mathbb{P}) = M' \cap \mathrm{Hb}(\mathbb{P})$. Now $w \leq \sum_{c \in C} w_c + \sum_{d \in D} w_d$, $D \cap N = \emptyset$, $C \subseteq N'$ and $h \notin N'$, which implies that there is a rule $r_3 = h \leftarrow C \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$ such that $N' \not\models r_3$, a contradiction, and therefore $M' \models r_2$.

Thus $M' \subset M$ and $M' \models P^M \cup \mathrm{F}_{M \cap I}$. This is contradictory to $M \in \mathrm{SM}(\mathbb{P})$, and we have $N = \mathrm{LM}(\mathrm{Tr}_{\mathrm{NLP}}(P)^N \cup \mathrm{F}_{N \cap I})$.

Define $f : \mathrm{SM}(\mathbb{P}) \to \mathrm{SM}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}))$ such that

$$f(M) = M \cup \{\overline{a} \mid \{H\} \leftarrow B^+, {\sim}B^- \in P \text{ and } a \in H \setminus M\}.$$

It is easy to see that $M \cap \mathrm{Hb_v}(\mathbb{P}) = f(M) \cap \mathrm{Hb_v}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}))$ and we need to show that $f$ is a bijection.

- $f$ is an injection: $M \neq M'$ implies $f(M) \neq f(M')$.
- $f$ is a surjection: consider arbitrary $N \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}))$. Let us show that $N \cap \mathrm{Hb}(\mathbb{P}) = M \in \mathrm{SM}(\mathbb{P})$ and $f(M) = f(N \cap \mathrm{Hb}(\mathbb{P})) = N$.

  (i) $M \models P^M \cup \mathrm{F}_{M \cap I}$:
  Clearly $M \models \mathrm{F}_{M \cap I}$. If $r_1 = h \leftarrow B^+ \in P^M$, then there is $\{H\} \leftarrow B^+, {\sim}B^- \in P$, such that $B^- \cap M = \emptyset$ and $h \in M \cap H$. Since $h \in M$, we have $M \models r_1$
  If $r_2 = h \leftarrow w' \leq \{B^+ = W_{B^+}\} \in P^M$ then there is a weight rule $h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \in P$ such that $w' = \max(0, w - \mathrm{WS}_M({\sim}B^- = W_{B^-}))$. Assume $M \not\models r_2$, that is, $h \notin M$ and $w' \leq \mathrm{WS}_M(B^+ = W_{B^+})$. Define $D = B^- \setminus M$ and $C = B^+ \cap M$. Since $N \cap \mathrm{Hb}(\mathbb{P}) = M$ and $w \leq \sum_{c \in C} w_c + \sum_{d \in D} w_d$, there is a rule $h \leftarrow C, {\sim}D \in \mathrm{Tr}_{\mathrm{NLP}}(P)$. Furthermore, $D \cap N = \emptyset$ implies $r_3 = h \leftarrow C \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$. Since $h \notin M$ implies $h \notin N$ and $C \subseteq M \subseteq N$, we have $N \not\models r_3$ which is a contradiction. Thus $M \models r_2$.

  (ii) $M = \mathrm{LM}(P^M \cup \mathrm{F}_{M \cap I})$:
  Assume there is $M' \subset M$ such that $M' \models P^M \cup \mathrm{F}_{M \cap I}$. Clearly $M' \cap I = M \cap I = N \cap I$. We define $N' = M' \cup (N \setminus \mathrm{Hb}(\mathbb{P})) \subset N$ and show $N' \models \mathrm{Tr}_{\mathrm{NLP}}(P)^N$. Since $N' \setminus \mathrm{Hb}(\mathbb{P}) = N \setminus \mathrm{Hb}(\mathbb{P})$, each rule of the form $\overline{a} \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$ is satisfied in $N'$. Also, each rule of the form $f_P \leftarrow b \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$ is satisfied in $N'$, as $N' \subset N$, $f_P \notin N$ and $N \models f_P \leftarrow b$.
  If $h \leftarrow C \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$ for $h \in \mathrm{Hb}(\mathbb{P})$, then either there is (a) a choice rule $\{H\} \leftarrow C, {\sim}D \in P$, such that $D \cap N = \emptyset$, $h \in H$ and $\overline{h} \notin N$ (which implies $h \in N$) or (b) a weight rule $h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \in P$ such that $w' = \max(0, w - \mathrm{WS}_N({\sim}B^- = W_{B^-}))$ and $w' = \sum_{c \in C} w_c$. Since $M \cap \mathrm{Hb}(\mathbb{P}) = N \cap \mathrm{Hb}(\mathbb{P})$, (a) implies that there is $h \leftarrow C \in P^M$ and (b) implies that there is $h \leftarrow w' \leq \{B^+ = W_{B^+}\} \in P^M$. Recalling $M' \cap \mathrm{Hb}(\mathbb{P}) = N' \cap \mathrm{Hb}(\mathbb{P})$ and $M' \models P^M$ we have $N' \models \mathrm{Tr}_{\mathrm{NLP}}(P)^N$. Thus $N' \models \mathrm{Tr}_{\mathrm{NLP}}(P)^N \cup \mathrm{F}_{N \cap I}$, which is a contradiction.

  (iii) $M \models \mathrm{CompS}(P)$:
  Assume that there is a compute statement compute $\{B^+, {\sim}B^-\}$ in $P$ that is not satisfied in $M$, that is, there is (a) $a \in B^+$ such that $a \notin M$ or (b) $b \in B^-$ such that $b \in M$. Recall that since $N$ is a stable model of $\mathrm{Tr}_{\mathrm{NLP}}(P)$, we have $f_P \notin N$. If (a), then $a \notin N$. Together with $f_P \notin N$ this implies that there is a fact $f_P \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$, a contradiction to $N \models \mathrm{Tr}_{\mathrm{NLP}}(P)^N$. If (b),

then $b \in N$. Since $f_P \notin N$, are is a rule $f_P \leftarrow b \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$. Thus $N \not\models f_P \leftarrow b$, again a contradiction to $N \models \mathrm{Tr}_{\mathrm{NLP}}(P)^N$. Thus $M \models \mathrm{CompS}(P)$.

(iv) $f(M) = f(N \cap \mathrm{Hb}(\mathbb{P})) = N$:
Define $N' = f(M) = f(N \cap \mathrm{Hb}(\mathbb{P}))$, that is,

$$N' = (N \cap \mathrm{Hb}(\mathbb{P})) \cup \{\overline{a} \mid \{H\} \leftarrow B^+, B^- \in P \text{ and } a \in H \setminus (N \cap \mathrm{Hb}(\mathbb{P})\}.$$

Now $N \cap \mathrm{Hb}(\mathbb{P}) = N' \cap \mathrm{Hb}(\mathbb{P})$. Since $N \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}))$, it holds $f_P \notin N$. Furthermore, $f_P \notin N'$ by definition. Assume that there is $\overline{a} \in N$ such that $\overline{a} \notin N'$. Since $\overline{a} \notin N'$, we have $a \in N'$ and furthermore $a \in N$. The only rule with $\overline{a}$ as the head is of the form $\overline{a} \leftarrow \sim a$. Thus, if $a \in N$, there is no rule with $\overline{a}$ as the head in $\mathrm{Tr}_{\mathrm{NLP}}(P)^N$ and therefore, since $N$ is a minimal model of $\mathrm{Tr}_{\mathrm{NLP}}(P)^N \cup \mathrm{F}_{N \cap I}$, we have $\overline{a} \notin N$, a contradiction.
On the other hand, assume that there is $\overline{a} \in N'$ such that $\overline{a} \notin N$. Since $\overline{a} \in N'$, we have $a \notin N'$ and furthermore $a \notin N$. If $a \notin N$, then there is a fact $\overline{a} \in \mathrm{Tr}_{\mathrm{NLP}}(P)^N$. Since $N \models \mathrm{Tr}_{\mathrm{NLP}}(P)^N$, we have $\overline{a} \in N$, a contradiction. Thus $N = N'$.

Thus $\mathbb{P} \equiv_{\mathrm{m}} \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P})$. □

**Proof of Theorem 5.17** Let $\mathbb{P}_1$ and $\mathbb{P}_2$ be SMODELS modules such that $\mathbb{P}_1 \sqcup \mathbb{P}_2$ is defined. By Theorem 5.15 also $\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2)$ is defined and applying Theorem 5.16 we get $\mathbb{P}_1 \sqcup \mathbb{P}_2 \equiv_{\mathrm{m}} \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1 \sqcup \mathbb{P}_2) \equiv_{\mathrm{m}} \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2)$. Therefore there is a bijection

$$f : \mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2) \rightarrow \mathrm{SM}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2))$$

such that for all $M \in \mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$ it holds $M_{\mathrm{v}} = (f(M))_{\mathrm{v}}$. As shown in the proof of Theorem 5.16, for any $M \in \mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$,

$$f(M) = M \cup \{\overline{a} \mid \{H\} \leftarrow B^+, \sim B^- \in P_1 \cup P_2 \text{ and } a \in H \setminus M\}.$$

Similarly by the proof of Theorem 5.16 we have bijections

$$f_1 : \mathrm{SM}(\mathbb{P}_1) \rightarrow \mathrm{SM}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1)) \text{ and } f_2 : \mathrm{SM}(\mathbb{P}_2) \rightarrow \mathrm{SM}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2));$$

with inverse functions $f_1^{-1}$ and $f_2^{-1}$ defined as

$$\begin{aligned} f_1^{-1}(N_1) &= N_1 \cap \mathrm{Hb}(\mathbb{P}_1) \in \mathrm{SM}(\mathbb{P}_1) \text{ for } N_1 \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1)) \text{ and} \\ f_2^{-1}(N_2) &= N_2 \cap \mathrm{Hb}(\mathbb{P}_2) \in \mathrm{SM}(\mathbb{P}_2) \text{ for } N_2 \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2)). \end{aligned}$$

Consider an arbitrary $M \in \mathrm{SM}(\mathbb{P}_1 \sqcup \mathbb{P}_2)$. We know that

$$f(M) \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2)).$$

Since $\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2)$ is a normal logic program module we can apply Theorem 3.9, and get $f(M) \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1) \sqcup \mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2))$ if and only if

$$\begin{aligned} N_1 &= f(M) \cap \mathrm{Hb}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1)) \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1)), \\ N_2 &= f(M) \cap \mathrm{Hb}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2)) \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2)), \end{aligned}$$

and $N_1$ and $N_2$ are compatible; or equivalently stated

$$\begin{array}{rcl} M_1 & = & f_1^{-1}(N_1) \\ & = & (f(M) \cap \mathrm{Hb}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_1))) \cap \mathrm{Hb}(\mathbb{P}_1) \\ & = & M \cap \mathrm{Hb}(\mathbb{P}_1) \in \mathrm{SM}(\mathbb{P}_1), \\ M_2 & = & f_2^{-1}(N_2) \\ & = & (f(M) \cap \mathrm{Hb}(\mathrm{Tr}_{\mathrm{NLP}}(\mathbb{P}_2))) \cap \mathrm{Hb}(\mathbb{P}_2) \\ & = & M \cap \mathrm{Hb}(\mathbb{P}_2) \in \mathrm{SM}(\mathbb{P}_2), \end{array}$$

and $M_1$ and $M_2$ are compatible. □

## A.4 PROOFS FOR CHAPTER 6

In this section, the proofs involve SMODELS *programs* and SMODELS *program modules* unless otherwise stated. Since basic rules and constraint rules are special cases of weight rules, we will assume that SMODELS programs and SMODELS program modules consist of choice rules, weight rules and compute statements only.

**Proof of Theorem 6.5** Let $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$ be SMODELS program modules having enough visible atoms and $\mathrm{EQT}(\mathbb{P}, \mathbb{Q})$ defined as in Definition 6.1. Consider a compatible collection $\{M, M_{\mathrm{H}}, M_{\mathrm{L}}, M_{\mathrm{U}}\}$ of interpretations for $\mathbb{P}$, $\mathrm{Hidden}^{\circ}(\mathbb{Q})$, $\mathrm{Least}^{\bullet}(\mathbb{Q})$, and $\mathrm{UnStable}(\mathbb{Q})$, respectively. By Corollary 5.18,

1. $M \in \mathrm{SM}(\mathbb{P})$,
2. $M_{\mathrm{H}} \in \mathrm{SM}(\mathrm{Hidden}^{\circ}(\mathbb{Q}))$,
3. $M_{\mathrm{L}} \in \mathrm{SM}(\mathrm{Least}^{\bullet}(\mathbb{Q}))$, and
4. $M_{\mathrm{U}} \in \mathrm{SM}(\mathrm{UnStable}(\mathbb{Q}))$

if and only if $M_{\mathrm{EQT}} = M \cup M_{\mathrm{H}} \cup M_{\mathrm{L}} \cup M_{\mathrm{U}} \in \mathrm{SM}(\mathrm{EQT}(\mathbb{P}, \mathbb{Q}))$.

Notice that there is no negative recursion between the modules (recall that positive recursion is not allowed in module composition), and therefore a compatible collection of stable models can be found sequentially starting from $M \in \mathrm{SM}(\mathbb{P})$. By Lemmas A.3–A.5 and Corollary 5.18, it holds $M_{\mathrm{EQT}} \in \mathrm{SM}(\mathrm{EQT}(\mathbb{P}, \mathbb{Q}))$ if and only if

- $M = M_{\mathrm{EQT}} \cap \mathrm{Hb}(\mathbb{P}) \in \mathrm{SM}(\mathbb{P})$,
- $N_{\mathrm{h}} = \mathrm{LM}((Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}})$,
- $K = \mathrm{LM}(Q^{N} \cup \mathrm{F}_{N \cap I})$ for $N = N_{\mathrm{h}} \cup M_{\mathrm{v}}$, and
- $K \neq N$ or $K \not\models \mathrm{CompS}(\mathbb{Q})$,

where $N_{\mathrm{h}}$ and $K$ can be extracted from $M_{\mathrm{H}}$ and $M_{\mathrm{L}}$ as done in Lemmas A.3–A.4. Each stable model of $\mathrm{EQT}(\mathbb{P}, \mathbb{Q})$ gives a counter-example for modular equivalence of $\mathbb{P}$ and $\mathbb{Q}$, and thus $\mathrm{SM}(\mathrm{EQT}(\mathbb{P}, \mathbb{Q})) = \mathrm{SM}(\mathrm{EQT}(\mathbb{Q}, \mathbb{P})) = \emptyset$ if and only if $\mathbb{P} \equiv_{\mathrm{m}} \mathbb{Q}$. □

**Lemma A.3** *Let* $\mathbb{P} = (P, I, O)$ *and* $\mathbb{Q} = (Q, I, O)$ *be* SMODELS *program modules having enough visible atoms. Consider an arbitrary* $M \in \mathrm{SM}(\mathbb{P})$ *and let* $\mathrm{Hidden}^{\circ}(\mathbb{Q})$ *be defined as in Definition 6.2. Let* $M_{\mathrm{H}} \subseteq \mathrm{Hb}(\mathrm{Hidden}^{\circ}(\mathbb{Q}))$ *and* $M$ *be compatible. Now,* $M_{\mathrm{H}} \in \mathrm{SM}(\mathrm{Hidden}^{\circ}(\mathbb{Q}))$ *if and only if* $N_{\mathrm{h}} = \mathrm{LM}((Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}})$*, where* $N_{\mathrm{h}} = \{a \in \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q}) \mid a^{\circ} \in M_{\mathrm{H}}\}$.

**Proof of Lemma A.3** Let $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$ be SMODELS program modules having enough visible atoms and consider an arbitrary $M \in \mathrm{SM}(\mathbb{P})$. Let $M_{\mathrm{H}} \subseteq \mathrm{Hb}(\mathrm{Hidden}^{\circ}(\mathbb{Q}))$ be compatible with $M$, that is,

$$M \cap (I \cup O) = M_{\mathrm{H}} \cap (I \cup O).$$

We define

$$N = N_\mathrm{v} \cup N_\mathrm{h} = (M_\mathrm{H} \cap \mathrm{Hb}_\mathrm{v}(\mathbb{Q})) \cup \{a \in \mathrm{Hb}_\mathrm{h}(\mathbb{Q}) \mid a^\circ \in M_\mathrm{H}\}.$$

Thus $N \subseteq \mathrm{Hb}(\mathbb{Q})$ is an interpretation for module $\mathbb{Q}$. Furthermore, since $M_\mathrm{H}$ is compatible with $M$, it holds $N_\mathrm{v} = M_\mathrm{v}$. The reduct $\mathrm{Hidden}^\circ(Q)^{M_\mathrm{H}}$ contains the following rules:

1. $h^\circ \leftarrow (B_\mathrm{h}^+)^\circ, B_\mathrm{v}^+$ if and only if there is a choice rule $\{H\} \leftarrow B^+, {\sim}B^-$ in $Q$ such that $h \in H_\mathrm{h}$, $h^\circ \in M_\mathrm{H}$, $(B_\mathrm{h}^-)^\circ \cap M_\mathrm{H} = \emptyset$, and $B_\mathrm{v}^- \cap M_\mathrm{H} = \emptyset$, or equivalently $h \in H_\mathrm{h} \cap N_\mathrm{h}$ and $B^- \cap N = \emptyset$; and

2. $h^\circ \leftarrow w' \leq \{(B_\mathrm{h}^+)^\circ \cup B_\mathrm{v}^+ = W_{B^+}\}$ if and only if there is a weight rule $h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \in Q$ such that $h \in \mathrm{Hb}_\mathrm{h}(\mathbb{Q})$, and $w' = \max(0, w - \mathrm{WS}_{M_\mathrm{H}}({\sim}((B_\mathrm{h}^-)^\circ \cup B_\mathrm{v}^-) = W_{B^-}))$, or equivalently $w' = \max(0, w - \mathrm{WS}_N({\sim}B^- = W_{B^-}))$.

($\Rightarrow$) Assume $M_\mathrm{H} \in \mathrm{SM}(\mathrm{Hidden}^\circ(\mathbb{Q}))$. First, we show $N_\mathrm{h} \models (Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}}$.

- If $r_1 = h \leftarrow B_\mathrm{h}^+ \in (Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}}$ then there is a choice rule $\{H\} \leftarrow B^+, {\sim}B^- \in Q$ such that $h \in H_\mathrm{h} \cap N_\mathrm{h}$, $B_\mathrm{v}^+ \subseteq M_\mathrm{v}$, $B_\mathrm{v}^- \cap M_\mathrm{v} = \emptyset$ and $B_\mathrm{h}^- \cap N_\mathrm{h} = \emptyset$. Since $h \in N_\mathrm{h}$, we have $N_\mathrm{h} \models r_1$ regardless of the satisfaction of $B_\mathrm{h}^+$ in $N_\mathrm{h}$.

- If $r_1 = h \leftarrow w_1 \leq \{B_\mathrm{h}^+ = W_{B_\mathrm{h}^+}\} \in (Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}}$ then there is a weight rule $h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \in Q$ such that $h \in \mathrm{Hb}_\mathrm{h}(\mathbb{Q})$ and
$$\begin{aligned} w_1 = \max(0, w - \mathrm{WS}_{M_\mathrm{v}}(B_\mathrm{v}^+ = W_{B_\mathrm{v}^+}, {\sim}B_\mathrm{v}^- = W_{B_\mathrm{v}^-}) \\ - \mathrm{WS}_{N_\mathrm{h}}({\sim}B_\mathrm{h}^- = W_{B_\mathrm{h}^-})). \end{aligned}$$

  Assume $N_\mathrm{h} \not\models r_1$, that is, $h \notin N_\mathrm{h}$ and $w_1 \leq \mathrm{WS}_{N_\mathrm{h}}(B_\mathrm{h}^+ = W_{B_\mathrm{h}^+})$. Using the definition of $w_1$ and recalling $N = M_\mathrm{v} \cup N_\mathrm{h}$ we get
$$w \leq \mathrm{WS}_N(B^+ = W_{B^+}, {\sim}B^- = W_{B^-}). \tag{A.6}$$

  Also, $r_2 = h^\circ \leftarrow w_2 \leq \{(B_\mathrm{h}^+)^\circ \cup B_\mathrm{v}^+ = W_{B^+}\} \in \mathrm{Hidden}^\circ(Q)^{M_\mathrm{H}}$ with $w_2 = \max(0, w - \mathrm{WS}_N({\sim}B^- = W_{B^-}))$. Since $M_\mathrm{H} = M_\mathrm{v} \cup N_\mathrm{h}^\circ$ we obtain
$$w_2 \;\leq\; \mathrm{WS}_N(B^+ = W_{B^+}) \;=\; \mathrm{WS}_{M_\mathrm{H}}((B_\mathrm{h}^+)^\circ \cup B_\mathrm{v}^+ = W_{B^+})$$

  from the definition of $w_2$ and (A.6). Furthermore $h^\circ \notin M_\mathrm{H}$. Thus $M_\mathrm{H} \not\models r_2$, a contradiction.

Thus $N_\mathrm{h} \models Q_\mathrm{h}/{M_\mathrm{v}}^{N_\mathrm{h}}$. Assume that $N_\mathrm{h} \neq \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})$, that is, there is $N_1 \subset N_\mathrm{h}$ such that $N_1 \models (Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}}$. Now, $M_\mathrm{v} \cup N_1^\circ \subset M_\mathrm{H}$. Clearly $M_\mathrm{v} \cup N_1^\circ \models \mathrm{F}_{M_\mathrm{H} \cap (I \cup O)}$. Also, $M_\mathrm{v} \cup N_1^\circ \models \mathrm{Hidden}^\circ(Q)^{M_\mathrm{H}}$ holds.

- If $r_1 = h^\circ \leftarrow (B_\mathrm{h}^+)^\circ, B_\mathrm{v}^+ \in \mathrm{Hidden}^\circ(Q)^{M_\mathrm{H}}$ then there is a choice rule $\{H\} \leftarrow B^+, {\sim}B^- \in Q$ such that $h \in H_\mathrm{h} \cap N_\mathrm{h}$ and $B^- \cap N = \emptyset$ (which implies $B_\mathrm{v}^- \cap M_\mathrm{v} = \emptyset$ and $B_\mathrm{h}^- \cap N_\mathrm{h} = \emptyset$). If $B_\mathrm{v}^+ \not\subseteq M_\mathrm{v}$, then $M_\mathrm{v} \cup N_1^\circ \models r_1$. On the other hand if $B_\mathrm{v}^+ \subseteq M_\mathrm{v}$, then there is $r_2 = h \leftarrow B_\mathrm{h}^+ \in (Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}}$. Since $N_1 \models r_2$ we have $M_\mathrm{v} \cup N_1^\circ \models r_1$.

- If $r_1 = h^\circ \leftarrow w_1 \leq \{(B^+_{\mathrm{h}})^\circ \cup B^+_{\mathrm{v}} = W_{B^+}\} \in \mathrm{Hidden}^\circ(Q)^{M_{\mathrm{H}}}$ then there is a weight rule $h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \in Q$ such that $h \in \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q})$ and $w_1 = \max(0, w - \mathrm{WS}_N({\sim}B^-_{\mathrm{h}} = W_{B^-}))$. Now, there is a rule $r_2 = h \leftarrow w_2 \leq \{B^+_{\mathrm{h}} = W_{B^+_{\mathrm{h}}}\} \in (Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}}$ such that

  $$w_2 = \max(0, w - \mathrm{WS}_{M_{\mathrm{v}}}(B^+_{\mathrm{v}} = W_{B^+_{\mathrm{v}}}) - \mathrm{WS}_N({\sim}B^- = W_{B^-})).$$

  This implies $w_2 = \max(0, w_1 - \mathrm{WS}_{M_{\mathrm{v}}}(B^+_{\mathrm{v}} = W_{B^+_{\mathrm{v}}}))$, and since $N_1 \models r_2$, it also holds $M_{\mathrm{v}} \cup N_1^\circ \models r_1$.

Thus $M_{\mathrm{v}} \cup N_1^\circ \models \mathrm{Hidden}^\circ(Q)^{M_{\mathrm{H}}} \cup \mathrm{F}_{M_{\mathrm{H}} \cap (I \cup O)}$, a contradiction to $M_{\mathrm{H}} = \mathrm{LM}(\mathrm{Hidden}^\circ(\mathbb{Q})^{M_{\mathrm{H}}} \cup \mathrm{F}_{M_{\mathrm{H}} \cap (I \cup O)})$. Thus $N_{\mathrm{h}} = \mathrm{LM}((Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}})$.

($\Leftarrow$) Assume $N_{\mathrm{h}} = \mathrm{LM}((Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}})$ and denote $N = M_{\mathrm{v}} \cup N_{\mathrm{h}}$. Now, $M_{\mathrm{H}} = M_{\mathrm{v}} \cup N_{\mathrm{h}}^\circ$ is compatible with $M$, and we need to show that $M_{\mathrm{H}} \in \mathrm{SM}(\mathrm{Hidden}^\circ(\mathbb{Q}))$, that is, $M_{\mathrm{H}} = \mathrm{LM}(\mathrm{Hidden}^\circ(Q)^{M_{\mathrm{H}}} \cup \mathrm{F}_{M_{\mathrm{H}} \cap (I \cup O)})$.

- If $r_1 = h^\circ \leftarrow (B^+_{\mathrm{h}})^\circ, B^+_{\mathrm{v}} \in \mathrm{Hidden}^\circ(Q)^{M_{\mathrm{H}}}$, there is a choice rule $\{H\} \leftarrow B^+, {\sim}B^- \in Q$ such that $h \in H_{\mathrm{h}} \cap N_{\mathrm{h}}$ and $B^- \cap N = \emptyset$. Now, $h \in N_{\mathrm{h}}$ implies $h^\circ \in M_{\mathrm{H}}$, and we have $M_{\mathrm{H}} \models r_1$ regardless of satisfaction of $(B^+_{\mathrm{h}})^\circ \cup B^+_{\mathrm{v}}$.

- If $r_1 = h^\circ \leftarrow w_1 \leq \{(B^+_{\mathrm{h}})^\circ \cup B^+_{\mathrm{v}} = W_{B^+}\} \in \mathrm{Hidden}^\circ(Q)^{M_{\mathrm{H}}}$ then there is a weight rule $h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \in Q$ such that $h \in \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q})$, and $w_1 = \max(0, w - \mathrm{WS}_N({\sim}B^- = W_{B^-}))$. Assume $M_{\mathrm{H}} \not\models r_1$, that is, $h^\circ \notin M_{\mathrm{H}}$ and $w_1 \leq \mathrm{WS}_{M_{\mathrm{H}}}((B^+_{\mathrm{h}})^\circ \cup B^+_{\mathrm{v}} = W_{B^+})$. Recalling $M_{\mathrm{H}} = M_{\mathrm{v}} \cup N_{\mathrm{h}}^\circ$ and the definition of $w_1$ we get

  $$w \leq \mathrm{WS}_{M_{\mathrm{v}}}(B^+_{\mathrm{v}} = W_{B^+_{\mathrm{v}}}) + \mathrm{WS}_{N_{\mathrm{h}}}(B^+_{\mathrm{h}} = W_{B^+_{\mathrm{h}}}) + \mathrm{WS}_N({\sim}B^- = W_{B^-}). \quad \text{(A.7)}$$

  On the other hand, $r_2 = h \leftarrow w_2 \leq \{B^+_{\mathrm{h}} = W_{B^+_{\mathrm{h}}}\} \in (Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}}$ for

  $$w_2 = \max(0, w - \mathrm{WS}_{M_{\mathrm{v}}}(B^+_{\mathrm{v}} = W_{B^+_{\mathrm{v}}}) - \mathrm{WS}_N({\sim}B^- = W_{B^-})).$$

  Together with (A.7) this implies $w_2 \leq \mathrm{WS}_{N_{\mathrm{h}}}(B^+_{\mathrm{h}} = W_{B^+_{\mathrm{h}}})$ and furthermore $N_{\mathrm{h}} \not\models r_2$, a contradiction to $N_{\mathrm{h}} \models (Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}}$. Thus $M_{\mathrm{H}} \models r_1$.

Since $M_{\mathrm{H}} \models r$ for each $r \in \mathrm{Hidden}^\circ(Q)^{M_{\mathrm{H}}}$, we have $M_{\mathrm{H}} \models \mathrm{Hidden}^\circ(Q)^{M_{\mathrm{H}}}$. Assume that $M_{\mathrm{H}} \neq \mathrm{LM}(\mathrm{Hidden}^\circ(Q)^{M_{\mathrm{H}}} \cup \mathrm{F}_{M_{\mathrm{H}} \cap (I \cup O)})$, that is, there is $M' \subset M_{\mathrm{H}}$ such that $M' \models \mathrm{Hidden}^\circ(Q)^{M_{\mathrm{H}}} \cup \mathrm{F}_{M_{\mathrm{H}} \cap (I \cup O)}$. We define

$$N' = \{a \mid a^\circ \in M'\} \subseteq \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q}).$$

Since $M' \models \mathrm{F}_{M_{\mathrm{H}} \cap (I \cup O)}$, we have $M' \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q}) = M_{\mathrm{H}} \cap \mathrm{Hb}_{\mathrm{v}}(\mathbb{Q}) = M_{\mathrm{v}}$ and furthermore $N' \subset N_{\mathrm{h}}$. Let us show that $N' \models (Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}}$.

- If $r_1 = h \leftarrow B^+_{\mathrm{h}} \in (Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}}$, then there is a choice rule $\{H\} \leftarrow B^+, {\sim}B^- \in Q$ such that $h \in N_{\mathrm{h}} \cap H_{\mathrm{h}}$, $B^+_{\mathrm{v}} \subseteq M_{\mathrm{v}}$, and $B^- \cap N = \emptyset$. Then also $r_2 = h^\circ \leftarrow (B^+_{\mathrm{h}})^\circ, B^+_{\mathrm{v}} \in \mathrm{Hidden}^\circ(Q)^{M_{\mathrm{H}}}$. Since $B^+_{\mathrm{v}} \subseteq M_{\mathrm{v}}$ implies $B^+_{\mathrm{v}} \subseteq M'$, and $M' \models r_2$, we have $N' \models r_1$.

- If $r_1 = h \leftarrow w_1 \leq \{B_{\mathrm{h}}^+ = W_{B_{\mathrm{h}}^+}\} \in (Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}}$ then there is a weight rule $h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \in Q$ such that $h \in \mathrm{Hb_h}(\mathbb{Q})$ and

  $$w_1 = \max(0, w - \mathrm{WS}_{M_{\mathrm{v}}}(B_{\mathrm{v}}^+ = W_{B_{\mathrm{v}}^+}) - \mathrm{WS}_N({\sim}B^- = W_{B^-})).$$

  Then $r_2 = h^\circ \leftarrow w_2 \leq \{(B_{\mathrm{h}}^+)^\circ \cup B_{\mathrm{v}}^+ = W_{B^+}\} \in \mathrm{Hidden}^\circ(Q)^{M_{\mathrm{H}}}$ with $w_2 = \max(0, w - \mathrm{WS}_N({\sim}B^- = W_{B^-}))$. Recall $M' = M_{\mathrm{v}} \cup (N')^\circ$. Now, $M' \models r_2$, that is, $h^\circ \notin M'$ or

  $$w \leq \mathrm{WS}_{N'}(B_{\mathrm{h}}^+ = W_{B_{\mathrm{h}}^+}) + \mathrm{WS}_{M_{\mathrm{v}}}(B_{\mathrm{v}}^+ = W_{B_{\mathrm{v}}^+}) + \mathrm{WS}_N({\sim}B^- = W_{B^-}). \quad \text{(A.8)}$$

  If $h^\circ \notin M'$, then $h \notin N'$ and $N' \models r_1$. Else if (A.8) holds, then by the definition of $w_1$ we have $w_1 \leq \mathrm{WS}_{N'}(B_{\mathrm{h}}^+ = W_{B_{\mathrm{h}}^+})$ and thus $N' \models r_1$.

Thus we have $N' \models (Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}}$, a contradiction to $N_{\mathrm{h}} = \mathrm{LM}((Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}})$, and therefore $M_{\mathrm{H}} \in \mathrm{SM}(\mathrm{Hidden}^\circ(\mathbb{Q}))$. □

**Lemma A.4** *Let $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$ be* SMODELS *program modules having enough visible atoms, and* $\mathrm{Least}^\bullet(\mathbb{Q})$ *defined as in Definition 6.3. Consider any compatible $M \in \mathrm{SM}(\mathbb{P})$, $M_{\mathrm{H}} \in \mathrm{SM}(\mathrm{Hidden}^\circ(\mathbb{Q}))$, and $M_{\mathrm{L}} \subseteq \mathrm{Hb}(\mathrm{Least}^\bullet(\mathbb{Q}))$. Now, $M_{\mathrm{L}} \in \mathrm{SM}(\mathrm{Least}^\bullet(\mathbb{Q}))$ if and only if $K = \mathrm{LM}(Q^N \cup \mathrm{F}_{N\cap I})$, where $K = \{a \in O \cup \mathrm{Hb_h}(\mathbb{Q}) \mid a^\bullet \in M_{\mathrm{L}}\} \cup (N \cap I)$ and $N = M_{\mathrm{v}} \cup N_{\mathrm{h}}$ for $N_{\mathrm{h}}$ defined as in Lemma A.3.*

**Proof of Lemma A.4** Let $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$ be SMODELS program modules having enough visible atoms, and $M \in \mathrm{SM}(\mathbb{P})$, $M_{\mathrm{H}} \in \mathrm{SM}(\mathrm{Hidden}^\circ(\mathbb{Q}))$, and $M_{\mathrm{L}} \subseteq \mathrm{Hb}(\mathrm{Least}^\bullet(\mathbb{Q}))$ be compatible. We define $J_{\mathrm{v}} = M_{\mathrm{L}} \cap \mathrm{Hb_v}(\mathbb{Q})$, $J_{\mathrm{h}} = \{a \in \mathrm{Hb_h}(\mathbb{Q}) \mid a^\circ \in M_{\mathrm{L}}\}$ and $L = \{a \in O \cup \mathrm{Hb_h}(\mathbb{Q}) \mid a^\bullet \in M_{\mathrm{L}}\}$. Thus $M_{\mathrm{L}} = J_{\mathrm{v}} \cup J_{\mathrm{h}}^\circ \cup L^\bullet$, and furthermore, $J = J_{\mathrm{v}} \cup J_{\mathrm{h}}$ is an interpretation for $\mathbb{Q}$ and $L \subseteq O \cup \mathrm{Hb_h}(\mathbb{Q})$. Since $M_{\mathrm{L}}$ is compatible with $M$ and $M_{\mathrm{H}}$, we have

$$\begin{aligned} M_{\mathrm{L}} \cap (I \cup O \cup \mathrm{Hb_h}(\mathbb{Q})^\circ) &= (M \cup M_{\mathrm{H}}) \cap (I \cup O \cup \mathrm{Hb_h}(\mathbb{Q})^\circ) \\ J_{\mathrm{v}} \cup J_{\mathrm{h}}^\circ &= M_{\mathrm{v}} \cup N_{\mathrm{h}}^\circ, \end{aligned}$$

which implies $J_{\mathrm{v}} = M_{\mathrm{v}}$ and $J_{\mathrm{h}} = N_{\mathrm{h}}$. The reduct $\mathrm{Least}^\bullet(Q)^{M_{\mathrm{L}}}$ contains the following rules:

1. $h^\bullet \leftarrow h, B_{\mathrm{i}}^+, (B_{\mathrm{o}}^+)^\bullet, (B_{\mathrm{h}}^+)^\bullet$ if and only if there is a choice rule $\{H\} \leftarrow B^+, {\sim}B^- \in Q$ such that $h \in H_{\mathrm{v}}$, $B_{\mathrm{v}}^- \cap M_{\mathrm{L}} = \emptyset$ and $(B_{\mathrm{h}}^-)^\circ \cap M_{\mathrm{L}} = \emptyset$, or equivalently $h \in H_{\mathrm{v}}$ and $B^- \cap N = \emptyset$;

2. $h^\bullet \leftarrow h^\circ, B_{\mathrm{i}}^+, (B_{\mathrm{o}}^+)^\bullet, (B_{\mathrm{h}}^+)^\bullet$ if and only if there is a choice rule $\{H\} \leftarrow B^+, {\sim}B^- \in Q$ such that $h \in H_{\mathrm{h}}$, $B_{\mathrm{v}}^- \cap M_{\mathrm{L}} = \emptyset$ and $(B_{\mathrm{h}}^-)^\circ \cap M_{\mathrm{L}} = \emptyset$, or equivalently $h \in H_{\mathrm{h}}$ and $B^- \cap N = \emptyset$; and

3. $h^\bullet \leftarrow w' \leq \{B_{\mathrm{i}}^+ \cup (B_{\mathrm{o}}^+)^\bullet \cup (B_{\mathrm{h}}^+)^\bullet = W_{B^+}\}$ if and only if there is a weight rule $h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \in Q$ such that $w' = \max(0, w - \mathrm{WS}_{M_{\mathrm{L}}}({\sim}(B_{\mathrm{v}}^- \cup (B_{\mathrm{h}}^-)^\circ) = W_{B^-}))$, or equivalently $w' = \max(0, w - \mathrm{WS}_N({\sim}B^- = W_{B^-}))$.

($\Rightarrow$) Assume $M_{\mathrm{L}} \in \mathrm{SM}(\mathrm{Least}^{\bullet}(\mathbb{Q}))$ and define $K = L \cup (N \cap I)$. Clearly $K \models \mathrm{F}_{N \cap I}$. We show that $K \models Q^N$.

- $r_1 = h \leftarrow B^+ \in Q^N$ if and only if there is choice rule $\{H\} \leftarrow B^+, {\sim}B^- \in Q$ such that $h \in H \cap N$ and $B^- \cap N = \emptyset$. Furthermore $h \in H \cap N$ implies $h \in H_{\mathrm{v}} \cap M_{\mathrm{v}}$ or $h^\circ \in H_{\mathrm{h}}^\circ \cap N_{\mathrm{h}}^\circ$. Thus we have $h \in M_{\mathrm{L}}$ or $h^\circ \in M_{\mathrm{L}}$, respectively.

  If $h \in H_{\mathrm{h}}$, then $r_2 = h^\bullet \leftarrow h^\circ, B_{\mathrm{i}}^+, (B_{\mathrm{o}}^+)^\bullet, (B_{\mathrm{h}}^+)^\bullet \in \mathrm{Least}^\bullet(Q)^{M_{\mathrm{L}}}$. Now $B_{\mathrm{i}}^+ \cup (B_{\mathrm{o}}^+)^\bullet \cup (B_{\mathrm{h}}^+)^\bullet \subseteq M_{\mathrm{L}}$ implies $h^\bullet \in M_{\mathrm{L}}$, since $M_{\mathrm{L}} \models r_2$ and $h^\circ \in M_{\mathrm{L}}$. Recalling the relation between $M_{\mathrm{L}}$ and $K$, we have also $K \models r_1$.

  If $h \in H_{\mathrm{v}}$, then $r_3 = h^\bullet \leftarrow h, B_{\mathrm{i}}^+, (B_{\mathrm{o}}^+)^\bullet, (B_{\mathrm{h}}^+)^\bullet \in \mathrm{Least}^\bullet(Q)^{M_{\mathrm{L}}}$. Now $B_{\mathrm{i}}^+ \cup (B_{\mathrm{o}}^+)^\bullet \cup (B_{\mathrm{h}}^+)^\bullet \subseteq M_{\mathrm{L}}$ implies $h^\bullet \in M_{\mathrm{L}}$, since $M_{\mathrm{L}} \models r_3$ and $h \in M_{\mathrm{L}}$. Again we have $K \models r_1$ by the relation between $M_{\mathrm{L}}$ and $K$.

- $r_1 = h \leftarrow w_1 \leq \{B^+ = W_{B^+}\} \in Q^N$ if and only if there is a weight rule $h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \in Q$ such that

  $$w_1 = \max(0, w - \mathrm{WS}_N({\sim}B^- = W_{B^-})).$$

  Then also, $r_2 = h^\bullet \leftarrow w_1 \leq \{B_{\mathrm{i}}^+ \cup (B_{\mathrm{o}}^+)^\bullet \cup (B_{\mathrm{h}}^+)^\bullet = W_{B^+}\} \in \mathrm{Least}^\bullet(Q)^{M_{\mathrm{L}}}$. Since $M_{\mathrm{L}} = M_{\mathrm{v}} \cup N_{\mathrm{h}}^\circ \cup L^\bullet$, $K = L \cup (N \cap I)$, and $M_{\mathrm{L}} \models r_2$, we have $K \models r_1$.

Thus $K \models Q^N$. Assume that $K \neq \mathrm{LM}(Q^N \cup \mathrm{F}_{N \cap I})$, that is, there is $K' \subset K$ such that $K' \models Q^N$ and $K' \cap I = K \cap I$. We define

$$M' = M_{\mathrm{v}} \cup N_{\mathrm{h}}^\circ \cup (K' \setminus I)^\bullet$$

and show that $M' \models \mathrm{Least}^\bullet(Q)^{M_{\mathrm{L}}}$.

- If $r_1 = h^\bullet \leftarrow h, B_{\mathrm{i}}^+, (B_{\mathrm{o}}^+)^\bullet, (B_{\mathrm{h}}^+)^\bullet \in \mathrm{Least}^\bullet(Q)^{M_{\mathrm{L}}}$ then there is a choice rule $\{H\} \leftarrow B^+, {\sim}B^- \in Q$ such that $h \in H_{\mathrm{v}}$, and $B^- \cap N = \emptyset$. If $h \notin M'$, then $M' \models r_1$. Else, we have $h \in M'$ which implies $h \in M_{\mathrm{v}}$, and there is $r_2 = h \leftarrow B^+ \in Q^N$. From the relation between $M'$ and $K'$ we see that $K' \models r_2$ implies $M' \models r_1$.

- If $r_1 = h^\bullet \leftarrow h^\circ, B_{\mathrm{i}}^+, (B_{\mathrm{o}}^+)^\bullet, (B_{\mathrm{h}}^+)^\bullet \in \mathrm{Least}^\bullet(Q)^{M_{\mathrm{L}}}$ then there is a choice rule $\{H\} \leftarrow B^+, {\sim}B^- \in Q$ such that $h \in H_{\mathrm{h}}$, $B^- \cap N = \emptyset$. If $h^\circ \notin M'$, then $M' \models r_1$. Else, we have $h^\circ \in M'$ which implies $h \in N_{\mathrm{h}}$, and there is $r_2 = h \leftarrow B^+ \in Q^N$. Again, we see from the relation between $M'$ and $K'$ that $K' \models r_2$ implies $M' \models r_1$.

- If $r_1 = h^\bullet \leftarrow w_1 \leq \{B_{\mathrm{i}}^+ \cup (B_{\mathrm{o}}^+)^\bullet \cup (B_{\mathrm{h}}^+)^\bullet = W_{B^+}\} \in \mathrm{Least}^\bullet(Q)^{M_{\mathrm{L}}}$ then there is a weight rule $h \leftarrow w \leq \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \in Q$ such that $w_1 = \max(0, w - \mathrm{WS}_N({\sim}B^- = W_{B^-}))$. Then also $r_2 = h \leftarrow w_1 \leq \{B^+ = W_{B^+}\} \in Q^N$. Now $K' \models r_2$ implies $M' \models r_1$.

Thus $M' \models \mathrm{Least}^\bullet(Q)^{M_{\mathrm{L}}} \cup \mathrm{F}_{M_{\mathrm{L}} \cap (I \cup O \cup \mathrm{Hb}_{\mathrm{h}}(\mathbb{Q})^\circ)}$, which is a contradiction to $M_{\mathrm{L}} \in \mathrm{SM}(\mathrm{Least}^\bullet(\mathbb{Q}))$. Therefore it holds $K = \mathrm{LM}(Q^N \cup \mathrm{F}_{N \cap I})$.

($\Leftarrow$) Assume that $K = \mathrm{LM}(Q^N \cup \mathrm{F}_{N\cap I})$, where $N = M_\mathrm{v} \cup N_\mathrm{h}$, $N_\mathrm{h} = \mathrm{LM}((\mathbb{Q}_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})$, and $M \in \mathrm{SM}(\mathbb{P})$. Now $M_\mathrm{L} = M_\mathrm{v} \cup N_\mathrm{h}^\circ \cup L^\bullet$ for $L = K \setminus I$ is compatible with $M$ and $M_\mathrm{H}$.

Clearly, $M_\mathrm{L} \models \mathrm{F}_{M_\mathrm{L}\cap(I\cup O\cup \mathrm{Hb}_\mathrm{h}(\mathbb{Q})^\circ)}$, and we show that $M_\mathrm{L} \models \mathrm{Least}^\bullet(Q)^{M_\mathrm{L}}$.

- If $r_1 = h^\bullet \leftarrow h, B_\mathrm{i}^+, (B_\mathrm{o}^+)^\bullet, (B_\mathrm{h}^+)^\bullet \in \mathrm{Least}^\bullet(Q)^{M_\mathrm{L}}$ then there is a choice rule $\{H\} \leftarrow B^+, {\sim}B^- \in Q$ such that $h \in H_\mathrm{v}$ and $B^- \cap N = \emptyset$. If $h \notin M_\mathrm{L}$, then $M_\mathrm{L} \models r_1$. Else, we have $h \in M_\mathrm{v}$ and there is $r_2 = h \leftarrow B^+ \in Q^N$. Since $K \models r_2$, we have $M_\mathrm{L} \models r_1$.

- If $r_1 = h^\bullet \leftarrow h^\circ, B_\mathrm{i}^+, (B_\mathrm{o}^+)^\bullet, (B_\mathrm{h}^+)^\bullet \in \mathrm{Least}^\bullet(Q)^{M_\mathrm{L}}$ then there is a choice rule $\{H\} \leftarrow B^+, {\sim}B^- \in Q$ such that $h \in H_\mathrm{h}$ and $B^- \cap N = \emptyset$. If $h^\circ \notin M_\mathrm{L}$, then $M_\mathrm{L} \models r_1$. Else $h \in N_\mathrm{h}$ and there is $r_2 = h \leftarrow B^+ \in Q^N$. Again, $K \models r_2$ implies $M_\mathrm{L} \models r_1$.

- If $r_1 = h^\bullet \leftarrow w_1 \le \{B_\mathrm{i}^+ \cup (B_\mathrm{o}^+)^\bullet \cup (B_\mathrm{h}^+)^\bullet = W_{B^+}\} \in \mathrm{Least}^\bullet(Q)^{M_\mathrm{L}}$ then there is a weight rule $h \leftarrow w \le \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \in Q$ such that $w_1 = \max(0, w - \mathrm{WS}_N({\sim}B^- = W_{B^-}))$. Then also $r_2 = h \leftarrow w_1 \le \{B^+ = W_{B^+}\} \in Q^N$. Since $K \models r_2$ holds, it also holds $M_\mathrm{L} \models r_1$.

Assume that $M_\mathrm{L} \neq \mathrm{LM}(\mathrm{Least}^\bullet(Q)^{M_\mathrm{L}} \cup \mathrm{F}_{M_\mathrm{L}\cap(I\cup O\cup \mathrm{Hb}_\mathrm{h}(\mathbb{Q})^\circ)})$, that is, there is $M' \subset M_\mathrm{L}$ such that $M' \models \mathrm{Least}^\bullet(Q)^{M_\mathrm{L}} \cup \mathrm{F}_{M_\mathrm{L}\cap(I\cup O\cup \mathrm{Hb}_\mathrm{h}(\mathbb{Q})^\circ)}$. Now,

$$M' \cap (I \cup O \cup \mathrm{Hb}_\mathrm{h}(\mathbb{Q})^\circ) = M_\mathrm{L} \cap (I \cup O \cup \mathrm{Hb}_\mathrm{h}(\mathbb{Q})^\circ).$$

We define $L' = \{a \mid a^\bullet \in M'\}$ and $K' = L' \cup (N \cap I)$. Now, $K' \subset K$ and we show in the following that $K' \models Q^N$.

- If $r_1 = h \leftarrow B^+ \in Q^N$ then there is a choice rule $\{H\} \leftarrow B^+, {\sim}B^- \in Q$ such that $h \in H \cap N$ and $B^- \cap N = \emptyset$. Now, $h \in H \cap N$ implies $h \in M_\mathrm{L}$ or $h^\circ \in M_\mathrm{L}$ for $h \in H_\mathrm{v}$ and $h \in H_\mathrm{h}$, respectively. Furthermore $h \in M_\mathrm{L}$ (respectively $h^\circ \in M_\mathrm{L}$) implies $h \in M'$ (respectively $h^\circ \in M'$).

  If $h \in H_\mathrm{h}$, then $r_2 = h^\bullet \leftarrow h^\circ, B_\mathrm{i}^+, (B_\mathrm{o}^+)^\bullet, (B_\mathrm{h}^+)^\bullet \in \mathrm{Least}^\bullet(Q)^{M_\mathrm{L}}$. Since $M' \models r_2$ and $h^\circ \in M'$, $B_\mathrm{i}^+ \cup (B_\mathrm{o}^+)^\bullet \cup (B_\mathrm{h}^+)^\bullet \subseteq M'$ implies $h^\bullet \in M'$. Recalling $M' = M_\mathrm{v} \cup N_\mathrm{h}^\circ \cup (L')^\bullet$ and $K' = L' \cup (N \cap I)$ we see that $M' \models r_2$ implies $K' \models r_1$.

  If $h \in H_\mathrm{v}$, then $r_3 = h^\bullet \leftarrow h, B_\mathrm{i}^+, (B_\mathrm{o}^+)^\bullet, (B_\mathrm{h}^+)^\bullet \in \mathrm{Least}^\bullet(Q)^{M_\mathrm{L}}$. Since $M' \models r_3$ and $h \in M'$, $B_\mathrm{i}^+ \cup (B_\mathrm{o}^+)^\bullet \cup (B_\mathrm{h}^+)^\bullet \subseteq M'$ implies $h^\bullet \in M'$. Again, we see that $M' \models r_3$ implies $K' \models r_1$.

- If $r_1 = h \leftarrow w_1 \le \{B^+ = W_{B^+}\} \in Q^N$ then there is a weight rule $h \leftarrow w \le \{B^+ = W_{B^+}, {\sim}B^- = W_{B^-}\} \in Q$ such that

  $$w_1 = \max(0, w - \mathrm{WS}_N({\sim}B^- = W_{B^-})).$$

  Then also, $r_2 = h^\bullet \leftarrow w_1 \le \{B_\mathrm{i}^+ \cup (B_\mathrm{o}^+)^\bullet \cup (B_\mathrm{h}^+)^\bullet = W_{B^+}\} \in \mathrm{Least}^\bullet(Q)^{M_\mathrm{L}}$ and $M' \models r_2$ implies $K' \models r_1$.

Thus $K' \models Q^N \cup \mathrm{F}_{N\cap I}$, a contradiction to $K = \mathrm{LM}(Q^N \cup \mathrm{F}_{N\cap I})$, and $M_\mathrm{L} \in \mathrm{SM}(\mathrm{Least}^\bullet(\mathbb{Q}))$ holds. □

**Lemma A.5** *Let $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$ be* SMODELS *program modules having enough visible atoms, and* $\mathrm{UnStable}(\mathbb{Q})$ *defined as in Definition 6.4. Consider arbitrary $M \in \mathrm{SM}(\mathbb{P})$, $M_{\mathrm{H}} \in \mathrm{SM}(\mathrm{Hidden}^{\circ}(\mathbb{Q}))$, $M_{\mathrm{L}} \in \mathrm{SM}(\mathrm{Least}^{\bullet}(\mathbb{Q}))$ and $M_{\mathrm{U}} \subseteq \mathrm{Hb}(\mathrm{UnStable}(\mathbb{Q}))$ such that the collection $\{M, M_{\mathrm{H}}, M_{\mathrm{L}}, M_{\mathrm{U}}\}$ is compatible. Now, $M_{\mathrm{U}} \in \mathrm{SM}(\mathrm{UnStable}(\mathbb{Q}))$ if and only if $N \neq K$ or $K \not\models \mathrm{CompS}(\mathbb{Q})$, where $K$ and $N$ are defined as in Lemmas A.3 and A.4, respectively.*

**Proof of Lemma A.5** Let $\mathbb{P} = (P, I, O)$ and $\mathbb{Q} = (Q, I, O)$ be SMODELS program modules having enough visible atoms and consider $M \in \mathrm{SM}(\mathbb{P})$, $M_{\mathrm{H}} \in \mathrm{SM}(\mathrm{Hidden}^{\circ}(\mathbb{Q}))$, $M_{\mathrm{L}} \in \mathrm{SM}(\mathrm{Least}^{\bullet}(\mathbb{Q}))$ and $M_{\mathrm{U}} \subseteq \mathrm{Hb}(\mathrm{UnStable}(\mathbb{Q}))$ such that the collection $\{M, M_{\mathrm{H}}, M_{\mathrm{L}}, M_{\mathrm{U}}\}$ is compatible. This implies

$$M_{\mathrm{U}} \setminus \{c, d, e\} = (M \cup M_{\mathrm{H}} \cup M_{\mathrm{L}}) \setminus \mathrm{Hb_h}(\mathbb{P}).$$

The reduct $\mathrm{UnStable}(Q)^{M_{\mathrm{U}}}$ contains the following rules:

1. $d \leftarrow a$ if and only if $a^{\bullet} \notin M_{\mathrm{U}}$ and $a \in O$;
2. $d \leftarrow a^{\bullet}$ if and only if $a \notin M_{\mathrm{U}}$ and $a \in O$;
3. $d \leftarrow a^{\circ}$ if and only if $a^{\bullet} \notin M_{\mathrm{U}}$ and $a \in \mathrm{Hb_h}(\mathbb{Q})$;
4. $d \leftarrow a^{\bullet}$ if and only if $a^{\circ} \notin M_{\mathrm{U}}$ and $a \in \mathrm{Hb_h}(\mathbb{Q})$;
5. $c$ if and only if $d \notin M_{\mathrm{U}}$, $a^{\bullet} \notin M_{\mathrm{U}}$ and $a \in \mathrm{CompS}(\mathbb{Q})$ such that $a \in O \cup \mathrm{Hb_h}(\mathbb{Q})$;
6. $c \leftarrow b^{\bullet}$ if and only if $d \notin M_{\mathrm{U}}$ and $\sim b \in \mathrm{CompS}(\mathbb{Q})$ such that $b \in O \cup \mathrm{Hb_h}(\mathbb{Q})$;
7. $c$ if and only if $d \notin M_{\mathrm{U}}$, $a \notin M_{\mathrm{U}}$ and $a \in \mathrm{CompS}(\mathbb{Q})$ such that $a \in I$;
8. $c \leftarrow b$ if and only if $d \notin M_{\mathrm{U}}$ and $\sim b \in \mathrm{CompS}(\mathbb{Q})$ such that $b \in I$; and
9. $e \leftarrow c$ and $e \leftarrow d$.

($\Rightarrow$) Assume $M_{\mathrm{U}} \in \mathrm{SM}(\mathrm{UnStable}(\mathbb{Q}))$. Since compute $\{e\} \in \mathrm{UnStable}(\mathbb{Q})$, we must have $e \in M_{\mathrm{U}}$. Since $e \in M_{\mathrm{U}}$ and the only rules having $e$ as the head are the ones in item 9. we must have $c \in M_{\mathrm{U}}$ or $d \in M_{\mathrm{U}}$. If $d \in M_{\mathrm{U}}$, then the rules in items 1.–4. imply that $N \neq K$. If $d \notin M_{\mathrm{U}}$ and $c \in M_{\mathrm{U}}$, then the rules in items 5.–8. imply that $K \not\models \mathrm{CompS}(\mathbb{Q})$.

($\Leftarrow$) Assume that $K = \mathrm{LM}(Q^{N} \cup \mathrm{F}_{N \cap I})$, where $N = M_{\mathrm{v}} \cup N_{\mathrm{h}}$, and $N_{\mathrm{h}} = \mathrm{LM}((Q_{\mathrm{h}}/M_{\mathrm{v}})^{N_{\mathrm{h}}})$. Define $L = K \setminus I$. If $N \neq K$, define $M_{\mathrm{U}} = M_{\mathrm{v}} \cup N_{\mathrm{h}}^{\circ} \cup L^{\bullet} \cup \{d, e\}$. Since $e \in M_{\mathrm{U}}$, $M_{\mathrm{U}} \models \mathrm{CompS}(\mathrm{UnStable}(\mathbb{Q}))$. Furthermore all the rules in $\mathrm{UnStable}(Q)^{M_{\mathrm{U}}}$ are satisfied. It is also clear that $M_{\mathrm{U}}$ is minimal with respect to input $M_{\mathrm{v}} \cup N_{\mathrm{h}}^{\circ} \cup L^{\bullet}$. On the other hand, if $K = N$ and $K \not\models \mathrm{CompS}(\mathbb{Q})$, define $M_{\mathrm{U}} = M_{\mathrm{v}} \cup N_{\mathrm{h}}^{\circ} \cup L^{\bullet} \cup \{c, e\}$. Since $e \in M_{\mathrm{U}}$, $M_{\mathrm{U}} \models \mathrm{CompS}(\mathrm{UnStable}(\mathbb{Q}))$. Furthermore all the rules in $\mathrm{UnStable}(Q)^{M_{\mathrm{U}}}$ are satisfied. Notice especially that rules in items 1.–4. are satisfied since $K = N$. Again, it is clear that $M_{\mathrm{U}}$ is minimal with respect to input $M_{\mathrm{v}} \cup N_{\mathrm{h}}^{\circ} \cup L^{\bullet}$, and $M_{\mathrm{U}} \in \mathrm{SM}(\mathrm{UnStable}(\mathbb{Q}))$. □

**Proof of Theorem 6.7** Let the assumptions in Theorem 6.7 hold for modules $\mathbb{P} = (P, I, O)$, $\mathbb{Q} = (Q, I, O)$ and $\mathbb{C} = (C, I_C, O_C)$. Since $\mathbb{P} \sqcup \mathbb{C}$ and $\mathbb{Q} \sqcup \mathbb{C}$ are defined, also $\mathrm{EQT}(\mathbb{P}, \mathbb{Q}) \sqcup \mathbb{C}$ and $\mathrm{EQT}(\mathbb{Q}, \mathbb{P}) \sqcup \mathbb{C}$ are defined.

Assume first that $\mathrm{SM}(\mathrm{EQT}(\mathbb{P}, \mathbb{Q}) \sqcup \mathbb{C}) \neq \emptyset$ or $\mathrm{SM}(\mathrm{EQT}(\mathbb{Q}, \mathbb{P}) \sqcup \mathbb{C}) \neq \emptyset$. We assume without loss of generality, that there is $M_\sqcup \in \mathrm{SM}(\mathrm{EQT}(\mathbb{P}, \mathbb{Q}) \sqcup \mathbb{C})$. By Theorem 5.17,

$$\begin{aligned} M_{\mathrm{EQT}} &= M_\sqcup \cap \mathrm{Hb}(\mathrm{EQT}(\mathbb{P}, \mathbb{Q})) \in \mathrm{SM}(\mathrm{EQT}(\mathbb{P}, \mathbb{Q})), \\ M_C &= M_\sqcup \cap \mathrm{Hb}(\mathbb{C}) \in \mathrm{SM}(\mathbb{C}), \end{aligned}$$

and $M_{\mathrm{EQT}}$ is compatible with $M_C$. Furthermore, as shown in the proof of Theorem 6.5, we can extract a counter-example for modular equivalence of $\mathbb{P}$ and $\mathbb{Q}$ from $M_{\mathrm{EQT}}$:

- $M = M_{\mathrm{EQT}} \cap \mathrm{Hb}(\mathbb{P}) \in \mathrm{SM}(\mathbb{P})$;
- $N_\mathrm{h} = \{a \in \mathrm{Hb}_\mathrm{h}(\mathbb{Q}) \mid a^\circ \in M_{\mathrm{EQT}}\}$ and $N_\mathrm{h} = \mathrm{LM}((Q_\mathrm{h}/M_\mathrm{v})^{N_\mathrm{h}})$;
- $K = \{a \in O \cup \mathrm{Hb}_\mathrm{h}(\mathbb{Q}) \mid a^\bullet \in M_{\mathrm{EQT}}\} \cup (N \cap I)$ for $N = M_\mathrm{v} \cup N_\mathrm{h}$, and $K = \mathrm{LM}(Q^N \cup \mathrm{F}_{N \cap I})$; and
- $K \neq N$ or $K \not\models \mathrm{CompS}(\mathbb{Q})$.

Since $M_{\mathrm{EQT}}$ and $M_C$ are compatible and $M_{\mathrm{EQT}} \cap \mathrm{Hb}_\mathrm{v}(\mathbb{C}) = M \cap \mathrm{Hb}_\mathrm{v}(\mathbb{C})$, also $M$ and $M_C$ are compatible. Furthermore, since $M \in \mathrm{SM}(\mathbb{P})$, $M_C \in \mathrm{SM}(\mathbb{C})$, and $M_C$ is compatible with $M$, we have by Theorem 5.17

$$M \cup M_C \in \mathrm{SM}(\mathbb{P} \sqcup \mathbb{C}).$$

Also $N$ and $M_C$ are compatible, because $M_\mathrm{v} = N_\mathrm{v}$. We define $N_\sqcup = N \cup M_C$, and assume that $N_\sqcup \in \mathrm{SM}(\mathbb{Q} \sqcup \mathbb{C})$. Then by Theorem 5.17 we get $N \in \mathrm{SM}(\mathbb{Q})$, a contradiction, since either $N \neq \mathrm{LM}(Q^N \cup \mathrm{F}_{N \cap I})$ or $N \not\models \mathrm{CompS}(\mathbb{Q})$. Thus $N_\sqcup \notin \mathrm{SM}(\mathbb{Q} \sqcup \mathbb{C})$. Recalling $(N_\sqcup)_\mathrm{v} = (M \cup M_C)_\mathrm{v}$, we have a counter-example for modular equivalence of $\mathbb{P} \sqcup \mathbb{C}$ and $\mathbb{Q} \sqcup \mathbb{C}$.

Else, assume that $\mathrm{SM}(\mathrm{EQT}(\mathbb{P}, \mathbb{Q}) \sqcup \mathbb{C}) = \mathrm{SM}(\mathrm{EQT}(\mathbb{Q}, \mathbb{P}) \sqcup \mathbb{C}) = \emptyset$. By Theorem 5.17, there is no compatible pair of stable models for $\mathrm{EQT}(\mathbb{P}, \mathbb{Q})$ and $\mathbb{C}$, and no compatible pair of stable models for $\mathrm{EQT}(\mathbb{Q}, \mathbb{P})$ and $\mathbb{C}$. As each stable model of the translations $\mathrm{EQT}(\mathbb{P}, \mathbb{Q})$ and $\mathrm{EQT}(\mathbb{Q}, \mathbb{P})$ represents a counter-example for $\mathbb{P} \equiv_\mathrm{m} \mathbb{Q}$ and there is no compatible stable model of $\mathbb{C}$ for any of the counter-examples, we have $\mathbb{P} \sqcup \mathbb{C} \equiv_\mathrm{m} \mathbb{Q} \sqcup \mathbb{C}$. □

# B BENCHMARK ENCODINGS

## B.1 QUEENS ENCODINGS

The LPARSE encodings for the modules $\mathbb{G}^n_x$, $\mathbb{G}^n_y$, $\mathbb{C}^n_1$ and $\mathbb{C}^n_2$ used in Section 7.2 to solve the $n$-queens problem are as follows.

- Module $\mathbb{G}^n_x$ generates a placement of queens row-by-row.

  ```
  #options -d none
  d(1..q).

  1 { q(X,Y): d(Y) } 1 :- d(X).
  ```

- Module $\mathbb{G}^n_y$ generates a placement of queens column-by-column.

  ```
  #options -d none
  d(1..q).

  1 { q(X,Y): d(X) } 1 :- d(Y).
  ```

- Module $\mathbb{C}^n_1$ is a part of the $n$-queens encoding in [54] checking that the placement of queens given as input is valid.

  ```
  #options -d none
  #external q(X,Y).
  d(1..q). q(X,Y) :- d(X;Y).  %% for grounding

  :- q(X,Y), q(X1,Y), X1 != X, d(X;X1;Y).
  :- q(X,Y), q(X,Y1), Y1 != Y, d(X;Y;Y1).
  :- q(X,Y), q(X1,Y1), X != X1, Y != Y1,
     abs(X-X1) == abs(Y-Y1), d(X;X1;Y;Y1).
  ```

- Module $\mathbb{C}^n_2$ is a variant of $\mathbb{C}^n_1$ in which symmetric rule instances have been removed.

  ```
  #options -d none
  #external q(X,Y).
  d(1..q). q(X,Y) :- d(X;Y).  %% for grounding

  :- q(X,Y), q(X1,Y), X1 < X, d(X;X1;Y).
  :- q(X,Y), q(X,Y1), Y1 < Y, d(X;Y;Y1).
  :- q(X,Y), q(X1,Y1), X != X1, Y != Y1,
     abs(X-X1) == abs(Y-Y1), X < X1, d(X;Y;X1;Y1).
  ```

## B.2 HAMILTONIAN CYCLE ENCODINGS

The LPARSE encodings for the modules $\mathbb{G}_i^n$, $\mathbb{R}^n$, $\mathbb{H}_j^n$ and $\mathbb{HR}^n$ used in Section 7.3 for finding a Hamiltonian cycle for different families of directed graphs are as follows.

- Module $\mathbb{G}_i^n$ for $i = 1, \ldots, 5$ generates different families of directed graphs.

```
#hide.
#show arc(X,Y).     %% output
#domain vtx(X;Y;Z).
vtx(1..n).

{ arc(X,Y) } :- vtx(X;Y).          %% all graphs

% :- arc(X,X).                     %% irreflexive
% arc(Y,X) :- arc(X,Y).            %% symmetric
% :- arc(Y,X), arc(X,Y).           %% asymmetric
% arc(Y,Z) :- arc(X,Y), arc(X,Z). %% Euclidean edge
                                   %% relation
```

- Module $\mathbb{H}_1^n$ selects edges to be taken into a Hamiltonian cycle candidate by insisting that each vertex is incident to exactly two edges in the cycle given a set of edges as input (similarly to the Hamiltonian cycle encoding in [65]). Module $\mathbb{H}_2^n$ is an optimized variant of $\mathbb{H}_1^n$.

```
#hide.
#external arc(X,Y).
#show arc(X,Y).       %% input
#show hc(X,Y).        %% output
#domain vtx(X;Y).
#options -d none.
vtx(1..n).

2 { hc(X,Y1): vtx(Y1), hc(Y1,X): vtx(Y1) } 2. %% (*)
:- hc(X,Y), not arc(X,Y).

% 1 { hc(X,Y1): vtx(Y1) } 1. %% in optimized version
% 1 { hc(Y1,X): vtx(Y1) } 1. %% for replacement of (*)
```

- Module $\mathbb{R}^n$ checks that each vertex is reachable from the starting vertex along the edges in the cycle given a candidate for Hamiltonian cycle as input

```
#hide.
#external hc(X,Y).
#show hc(X,Y).        %% input
#show reached(X).     %% output
#domain vtx(X;Y).
#options -d none.
vtx(1..n).
{ hc(X,Y) }.          %% for grounding

initialvtx(1).
reached(Y) :- hc(X,Y), reached(X), not initialvtx(X).
reached(Y) :- hc(X,Y), initialvtx(X).
:- not reached(X).
```

- Module $\mathbb{HR}^n$ solves the Hamiltonian cycle problem given a set of edges as input (based on the alternative encoding presented in [65]).

```
#hide.
#external arc(X,Y).
#show arc(X,Y).            %% input
#show hc(X,Y), reached(X). %% output
#domain vtx(X;Y;X1;Y1).
#options -d none.
vtx(1..n).
{ arc(X,Y) }.              %% for grounding

start(1).
{ hc(X,Y) } :- start(X), arc(X,Y).
{ hc(X,Y) } :- reached(X), arc(X,Y).

reached(Y) :- hc(X,Y).
:- not reached(X).
:- hc(X,Y), hc(X,Y1), Y != Y1.
:- hc(X,Y), hc(X1,Y), X != X1.
```

# BIBLIOGRAPHY

[1] Christian Anger, Martin Gebser, Thomas Linke, André Neumann, and Torsten Schaub. The nomore++ system. In Baral et al. [3], pages 422–426.

[2] Marcello Balduccini, Michael Gelfond, Richard Watson, and Monica Nogueira. The USA-advisor: A case study in answer set planning. In Thomas Eiter, Wolfgang Faber, and Miroslaw Truszczyński, editors, *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 2173 of *Lecture Notes in Computer Science*, pages 439–442, Vienna, Austria, September 2001. Springer.

[3] Chitta Baral, Gianluigi Greco, Nicola Leone, and Giorgio Terracina, editors. *Logic Programming and Nonmonotonic Reasoning, 8th International Conference, LPNMR 2005, Diamante, Italy, September 5-8, 2005, Proceedings*, volume 3662 of *Lecture Notes in Computer Science*. Springer, 2005.

[4] Antonio Brogi, Paolo Mancarella, Dino Pedreschi, and Franco Turini. Modular logic programming. *ACM Transactions on Programming Languages and Systems*, 16(4):1361–1398, 1994.

[5] Michele Bugliesi, Evelina Lamma, and Paola Mello. Modularity in logic programming. *Journal of Logic Programming*, 19/20:443–502, 1994.

[6] Pawel Cholewinski and Miroslaw Truszczynski. Extremal problems in logic programming and stable model computation. *Journal of Logic Programming*, 38(2):219–242, 1999.

[7] Thomas Eiter and Michael Fink. Uniform equivalence of logic programs under the stable model semantics. In Catuscia Palamidessi, editor, *Proceedings of the 19th International Conference on Logic Programming*, volume 2916 of *Lecture Notes in Computer Science*, pages 224–238, Mumbay, India, December 2003. Springer.

[8] Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. Simplifying logic programs under uniform and strong equivalence. In Lifschitz and Niemel [36], pages 87–99.

[9] Thomas Eiter, Michael Fink, and Stefan Woltran. Semantical characterizations and complexity of equivalences in answer set programming. *ACM Transactions on Computational Logic*, To appear, see `http://www.acm.org/pubs/tocl/accepted.html`.

[10] Thomas Eiter, Georg Gottlob, and Heikki Mannila. Disjunctive datalog. *ACM Transactions on Database Systems*, 22(3):364–418, 1997.

[11] Thomas Eiter, Georg Gottlob, and Helmut Veith. Modular logic programming and generalized quantifiers. In J rgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1265 of *Lecture Notes in Computer Science*, pages 290–309. Springer, 1997.

[12] Thomas Eiter, Hans Tompits, and Stefan Woltran. On solution correspondences in answer-set programming. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence*, pages 97–102, Edinburgh, Scotland, UK, July/August 2005. Professional Book Center.

[13] Esra Erdem, Vladimir Lifschitz, and Martin D. F. Wong. Wire routing and satisfiability planning. In John W. Lloyd, Verónica Dahl, Ulrich Furbach, Manfred Kerber, Kung-Kiu Lau, Catuscia Palamidessi, Lu s Moniz Pereira, Yehoshua Sagiv, and Peter J. Stuckey, editors, *Proceedings of the First International Conference on Computational Logic, Automated Deduction: Putting Theory into Practice*, volume 1861 of *Lecture Notes in Computer Science*, pages 822–836, London, UK, July 2000. Springer.

[14] Sandro Etalle and Maurizio Gabbrielli. Transformations of CLP modules. *Theoretical Computer Science*, 166(1&2):101–146, 1996.

[15] Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Magic sets and their application to data integration. In Thomas Eiter and Leonid Libkin, editors, *Proceedings of the 10th International Conference on Database Theory*, volume 3363 of *Lecture Notes in Computer Science*, pages 306–320, Edinburgh, UK, January 2005. Springer.

[16] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. *Theory and Practice of Logic Programming*, 5(1-2):45–74, 2005.

[17] Haim Gaifman and Ehud Y. Shapiro. Fully abstract compositional semantics for logic programs. In *Proceedings of the 16th Annual ACM Symposium on Principles of Programming Languages*, pages 134–142, Austin, Texas, USA, January 1989. ACM Press.

[18] Martin Gebser, Benjamin Kaufmann, Andre Neumann, and Torsten Schaub. Conflict-driven answer set solving. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI'07)*, 2007. To Appear.

[19] Michael Gelfond and Nicola Leone. Logic programming and knowledge representation - the A-Prolog perspective. *Artificial Intelligence*, 138(1-2):3–38, 2002.

[20] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Kowalski and Bowen [32], pages 1070–1080.

[21] Michael Gelfond and Vladimir Lifschitz. Logic programs with classical negation. In David H. D. Warren and Péter Szeredi, editors, *Logic programming, Proceedings of the Seventh International Conference, Jerusalem, Israel, June 18-20, 1990*, pages 579–597, Cambridge, MA, USA, 1990. MIT Press.

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

[23] Laura Giordano and Alberto Martelli. Structuring logic programs: A modal approach. *Journal of Logic Programming*, 21(2):59–94, 1994.

[24] 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.

[25] Maarit Hietalahti, Fabio Massacci, and Ilkka Niemel . 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.

[26] Giovambattista Ianni, Giuseppe Ielpa, Adriana Pietramala, Maria Carmela Santoro, and Francesco Calimeri. Enhancing answer set programming with templates. In James P. Delgrande and Torsten Schaub, editors, *10th International Workshop on Non-Monotonic Reasoning (NMR 2004), Whistler, Canada, June 6-8, 2004, Proceedings*, pages 233–239, 2004.

[27] Joxan Jaffar and Michael J. Maher. Constraint logic programming: A survey. *Journal of Logic Programming*, 19/20:503–581, 1994.

[28] Tomi Janhunen. Translatability and intranslatability results for certain classes of logic programs. Research Report A82, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, November 2003.

[29] Tomi Janhunen, Ilkka Niemel , Dietmar Seipel, Patrik Simons, and Jia-Huai You. Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computational Logic*, 7(1):1–37, 2006.

[30] 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, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence*, volume 2424 of *Lecture Notes in Computer Science*, pages 493–504, Cosenza, Italy, September 2002. Springer.

[31] Tomi Janhunen and Emilia Oikarinen. Automated verification of weak equivalence within the SMODELS system. *Theory and Practice of Logic Programming*, to appear. Available at `http://arxiv.org/abs/cs/0608099`.

[32] Robert A. Kowalski and Kenneth A. Bowen, editors. *Logic Programming, Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, August 15-19, 1988*. MIT Press, 1988.

[33] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 7(3):499–562, 2006.

[34] Yuliya Lierler and Marco Maratea. Cmodels-2: SAT-based answer set solver enhanced to non-tight programs. In Lifschitz and Niemel [36], pages 346–350.

[35] 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, November/December 1999. MIT Press.

[36] Vladimir Lifschitz and Ilkka Niemel , editors. *Logic Programming and Nonmonotonic Reasoning, 7th International Conference, LPNMR 2004, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings*, volume 2923 of *Lecture Notes in Computer Science*. Springer, 2004.

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

[38] 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.

[39] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Proceedings of the Eleventh International Conference on Logic Programming*, pages 23–37, Santa Margherita Ligure, Italy, June 1994. MIT Press.

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

[41] Fangzhen Lin and Yuting Zhao. ASSAT: Computing answer sets of a logic program by SAT solvers. *Artificial Intelligence*, 157(1-2):115–137, 2004.

[42] Zhijun Lin, Yuanlin Zhang, and Hector Hernandez. Fast SAT-based answer set solver. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*. AAAI Press, 2006.

[43] Per Lindstr m. First order predicate logic with generalized quantifiers. *Theoria*, 32:186–195, 1966.

[44] Lengning Liu and Miroslaw Truszczyński. Pbmodels - software to compute stable models by pseudoboolean solvers. In Baral et al. [3], pages 410–415.

[45] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, Berlin, 1987.

[46] Michael J. Maher. Equivalences of logic programs. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 627–658. Morgan Kaufmann, 1988.

[47] Michael J. Maher. A transformation system for deductive database modules with perfect model semantics. *Theoretical Computer Science*, 110(2):377–403, 1993.

[48] Paolo Mancarella and Dino Pedreschi. An algebra of logic programs. In Kowalski and Bowen [32], pages 1006–1023.

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

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

[51] Albert R. Meyer. Semantical paradigms: Notes for an invited lecture, with two appendices by Stavros S. Cosmadakis. In Yuri Gurevich, editor, *Proceedings, Third Annual Symposium on Logic in Computer Science, 5-8 July 1988, Edinburgh, Scotland, UK*, pages 236–253. IEEE Computer Society, 1988.

[52] Dale Miller. A theory of modules for logic programming. In Robert M. Keller, editor, *Proceedings of 1986 Symposium on Logic Programming*, pages 106–114, Salt Lake City, USA, September 1986. IEEE Computer Society Press.

[53] Andrzej Mostowski. On a generalization of quantifiers. *Fundamenta Mathematicae*, 44:12–36, 1957.

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

[55] Emilia Oikarinen. *Modular Answer Set Programming.* Licentiate's thesis, Helsinki University of Technology, Department of Computer Science and Engineering, Laboratory for Theoretical Computer Science, Espoo, Finland, November 2006.

[56] Emilia Oikarinen and Tomi Janhunen. Verifying the equivalence of logic programs in the disjunctive case. In Lifschitz and Niemel  [36], pages 180–193.

[57] Emilia Oikarinen and Tomi Janhunen. Modular equivalence for normal logic programs. In J ergen Dix and Anthony Hunter, editors, *Proceedings of the 11th International Workshop on Nonmonotonic Reasoning*, pages 10–18, Lake District, UK, May 2006. University of Clausthal, Department of Informatics, Technical Report, IfI-06-04.

[58] Emilia Oikarinen and Tomi Janhunen. Modular equivalence for normal logic programs. In Gerhard Brewka, Silvia Coradeschi, Anna Perini, and Paolo Traverso, editors, *Proceedings of the 17th European Conference on Artificial Intelligence*, pages 412–416, Riva del Garda, Italy, August 2006. IOS Press.

[59] Richard A. O'Keefe. Towards an algebra for constructing logic programs. In *Proceedings of the 1985 Symposium on Logic Programming*, pages 152–160, Boston, Massachusetts, USA, July 1985.

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

[61] David Pearce, Hans Tompits, and Stefan Woltran. Encodings for equilibrium logic and logic programs with nested expressions. In Pavel Brazdil and Al pio Jorge, editors, *Progress in Artificial Intelligence, Knowledge Extraction, Multi-agent Systems, Logic Programming and Constraint Solving, Proceedings of 10th Portuguese Conference on Artificial Intelligence, EPIA'01*, volume 2258 of *Lecture Notes in Computer Science*, pages 306–320, Porto, Portugal, December 2001. Springer.

[62] 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.

[63] Teodor C. Przymusinski. Perfect model semantics. In Kowalski and Bowen [32], pages 1081–1096.

[64] Yehoshua Sagiv. Optimizing datalog programs. In *Proceedings of the 6th ACM Symposium on Principles of database systems*, pages 349–362, San Diego, CA, USA, March 1987. ACM Press.

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

[66] Timo Soininen and Ilkka Niemel . 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*, volume 1551 of *Lecture Notes in Com-*

*puter Science*, pages 305–319, San Antonio, Texas, USA, January 1999. Springer.

[67] Tommi Syrj nen. Lparse 1.0 user's manual. `http://www.tcs.hut.fi/Software/smodels`, April 2001.

[68] Luis Tari, Chitta Baral, and Saadat Anwar. A language for modular answer set programming: Application to ACC tournament scheduling. In Marina De Vos and Alessandro Provetti, editors, *Proceedings of the 3rd International Workshop on Answer Set Programming*, volume 142 of *CEUR Workshop Proceedings*, Bath, UK, September 2005. CEUR-WS.org.

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

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

[71] Stefan Woltran. Characterizations for relativized notions of equivalence in answer set programming. In José Júlio Alferes and Jo o Alexandre Leite, editors, *Proceedings of the 9th European Conference on Logics in Artificial Intelligence*, volume 3229 of *Lecture Notes in Computer Science*, pages 161–173, Lisbon, Portugal, September 2004. Springer.

ISBN-13 978-951-22-8581-5
ISBN-10 951-22-8581-9
ISSN 1457-7615