Some (in)translatability results for normal logic programs and propositional theories

Tomi Janhunen

Helsinki University of Technology
Department of Computer Science and Engineering
P.O. Box 5400
FI-02015 TKK (Finland)
Tomi.Janhunen@tkk.fi

ABSTRACT. *In this article, we compare the expressive powers of classes of normal logic programs that are obtained by constraining the number of positive subgoals (n) in the bodies of rules. The comparison is based on the existence/nonexistence of polynomial, faithful, and modular (PFM) translation functions between the classes. As a result, we obtain a strict ordering among the classes under consideration. Binary programs ($n \leq 2$) are shown to be as expressive as unconstrained programs but strictly more expressive than unary programs ($n \leq 1$) which, in turn, are strictly more expressive than atomic programs ($n = 0$). We also take propositional theories into consideration and prove them to be strictly less expressive than atomic programs. In spite of the gap in expressiveness, we develop a faithful but non-modular translation function from normal programs to propositional theories. We consider this as a breakthrough due to sub-quadratic time complexity (of the order of $||P|| \times \log_2 |\mathrm{Hb}(P)|$). Furthermore, we present a prototype implementation of the translation function and demonstrate its promising performance with SAT solvers using three benchmark problems.*

KEYWORDS: *modularity, translation functions, expressive power, stable model semantics, answer set programming, satisfiability solvers.*

1. Introduction

Normal logic programs under the stable model semantics [GEL 88] are well-suited for a variety of knowledge representation tasks. Typically, a programmer solves a problem at hand (i) by formulating it as a logic program whose stable models correspond to the solutions of the problem and (ii) by computing stable models (or *answer sets* [GEL 90]) for the program using a special-purpose search engine. The reader is referred e.g. to [MAR 99, NIE 99, GEL 02] for examples of using this kind of constraint programming paradigm, also known as *answer set programming* (ASP). The

paradigm differs from conventional logic programming [EMD 76, LLO 87] which is based on query evaluation using the resolution principle [ROB 65]. The success of stable models and answer set programming is much due to efficient implementations, such as SMODELS [SIM 02] and DLV [EIT 98], which were developed in the 90s.

A basic approach to computing stable models is to use a *branch and bound* search algorithm [SUB 95] and the *well-founded model* [VAN 91] as an approximation in order to prune search space. Even tighter approximations can be employed given some assumptions (truth values of atoms) on stable models to be computed [SIM 02]. One particular approximation technique tries to apply rules *contrapositively*: if the *head* h of a rule $\mathsf{h} \leftarrow \mathsf{a}_1, \ldots, \mathsf{a}_n$ is known/assumed to be false in a stable model M being computed, then one of the positive subgoals a_i in the *body* must also be false in M. This becomes particularly effective when $n = 1$ or all other atoms except a_1 are known/assumed to be true in M. Then a_1 can be inferred to be false in M which refines the approximation a bit. These observations lead to the basic question that initiated our research: *Is it possible to reduce the number of positive subgoals appearing in rules in order to accelerate contrapositive reasoning?*

To address this problem, we analyze the expressiveness of classes of logic programs that are obtained by restricting the number of positive subgoals in the bodies of rules. The aim is to develop and apply a method that is similar to the one developed for non-monotonic logics [JAN 99b, JAN 03a]. According to this method, a basic step is to check the existence of a *polynomial, faithful and modular* (PFM) translation functions between classes. If such a translation function turns out to be non-existent, then the syntactic constraints imposed on the classes are significant and affect expressive power. In particular, if there is no PFM translation function that reduces the number of positive subgoals, then the reduction is likely to be infeasible in practice as it cannot be done in a localized (modular) way. The results of the expressiveness analysis indicate that the number of positive subgoals in a rule can be reduced down to two, but going below that limit is impossible in a faithful and modular way. In spite of this negative result, we manage to develop a faithful and *non-modular*, but still fairly systematic, translation function $\mathrm{Tr}_{\mathrm{AT}}$ for removing positive subgoals altogether.

The elimination of positive subgoals is also interesting as concerns propositional logic: by composing $\mathrm{Tr}_{\mathrm{AT}}$ with Clark's completion procedure [CLA 78] we obtain a novel reduction from normal logic programs to propositional theories. This opens new prospects as regards implementing ASP using SAT solvers for actual computations: a program can be reduced completely before computing models for the resulting propositional theory. This is in contrast with earlier approaches [LIN 04, LIE 04] that introduce *loop formulas* and extend the completion of the program incrementally while computing models. Moreover, by analyzing and composing reductions from ASP to SAT we gain new insight into their relationship. In fact, many problems that have been solved using ASP have also formulations as classical satisfiability problems. But such formulations tend to be more tedious to accomplish and less concise. For instance, formulating an AI planning problem is much easier as a normal logic program [DIM 97] than as a set of clauses [KAU 96]. Such practical experiences suggest a

real difference in expressive power which has already been demonstrated in terms of formal counter-examples [NIE 99].

These observations on the interconnection of ASP and satisfiability checking led to a further objective for our research. The goal is to develop efficient reductions from ASP to propositional theories and thus combine the knowledge representation capabilities of ASP with the efficiency of SAT solvers so that we can benefit from their rapid development. In this article, we approach this goal in the following respects. First, propositional theories are taken into account in the expressiveness analysis to better understand their relationship with the classes of normal logic programs under consideration. Second, a concrete reduction from ASP to propositional satisfiability is developed following the lines discussed above. Third, preliminary experiments are conducted using a prototype implementation and existing solvers. We aim at a proof of concept by comparing the performance of our translation-based approach with others.

The rest of this article is organized as follows. In Section 2, we review the syntax and semantics of the formalisms of our interest: normal logic programs and propositional theories. Moreover, we distinguish syntactic subclasses of normal logic programs to be analyzed in the sequel. Three central properties of translation functions are distinguished in Section 3: polynomiality, faithfulness, and modularity. Consequently, a method for comparing the expressive powers of classes of logic programs is established. The actual expressiveness analysis takes place in Section 4. The classes of logic programs are ordered on the basis of their expressive powers which gives rise to an expressive power hierarchy for the classes under consideration. These comparisons involve intranslatability results that count on the modularity property. This is why we pursue non-modular translation functions in Section 6. A particular objective in this respect is to translate faithfully normal logic programs into propositional theories in sub-quadratic time (The resulting translation function is based an alternative characterization of stable models devised earlier in Section 5.) After that related work is addressed in Section 7. We present a prototype implementation of the translation in Section 8 and compare its performance with a variety of answer set solvers using three benchmark problems. Finally, we present our conclusions in Section 9.

2. Preliminaries

In this section, we review the basic terminology and definitions of normal logic programs as well as propositional theories.

2.1. *Normal logic programs*

In this article, we restrict ourselves to the purely propositional case and consider only programs that consist of propositional atoms[1]. A *normal (logic) program* P is a set of *rules* which are expressions of the form

1. Programs with variables, constants and function symbols are encompassed implicitly through Herbrand instantiation. However, the forthcoming expressiveness analysis is based

$$\mathsf{a} \leftarrow \mathsf{b}_1, \ldots, \mathsf{b}_n, {\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m \tag{1}$$

where a is an atom, and $\{\mathsf{b}_1, \ldots, \mathsf{b}_n\}$ and $\{\mathsf{c}_1, \ldots, \mathsf{c}_m\}$ form sets of atoms. Here $\sim$ denotes *default negation* or Clark's *negation as failure to prove* [CLA 78], which differs from *classical negation* denoted by $\neg$. We define *default literals* in the standard way using $\sim$ as negation, and given a set of atoms A, we let ${\sim}A$ denote the set of negative literals $\{{\sim}\mathsf{a} \mid \mathsf{a} \in A\}$. Intuitively speaking, a rule r of the form (1) is used as follows: the *head* $\mathrm{H}(r) = \mathsf{a}$ of r can be inferred by applying r whenever the conditions in the body $\mathrm{B}(r) = \{\mathsf{b}_1, \ldots, \mathsf{b}_n\} \cup \{{\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m\}$ of r are met. This is the case when the *positive body atoms* in $\mathrm{B}^+(r) = \{\mathsf{b}_1, \ldots, \mathsf{b}_n\}$ are inferable by the rules of the program, but not the *negative body atoms* in $\mathrm{B}^-(r) = \{\mathsf{c}_1, \ldots, \mathsf{c}_m\}$. These intuitions will be made exact by model theory presented in Section 2.2.

In the sequel, the class of normal programs is denoted by $\mathcal{P}$. We extend the preceding notations to cover any $P \in \mathcal{P}$. For instance, $\mathrm{H}(P) = \{\mathrm{H}(r) \mid r \in P\}$, and $\mathrm{B}(P)$, $\mathrm{B}^+(P)$, and $\mathrm{B}^-(P)$ are analogously defined. The positive part r^+ of a rule r is defined as $\mathrm{H}(r) \leftarrow \mathrm{B}^+(r)$. A (normal) program P is *positive*, if $r = r^+$ holds for all rules $r \in P$. In addition to positive programs, we distinguish normal programs that are obtained by restricting the number of positive body atoms allowed in a rule r.

DEFINITION 1. — *A rule r of a normal program is called **atomic**, **unary** or **binary**, if $|\mathrm{B}^+(r)| = 0$, $|\mathrm{B}^+(r)| \leq 1$, or $|\mathrm{B}^+(r)| \leq 2$, respectively.*

Moreover, we say that a rule r is *strictly* unary if $|\mathrm{B}^+(r)| = 1$, i.e. it is unary and not atomic. *Strictly* binary rules are defined analogously, i.e. $|\mathrm{B}^+(r)| = 2$. We extend these conditions to cover a normal program P in the obvious way: P is atomic, unary, or binary if every rule of P satisfies the respective condition. E.g., an *atomic normal program* P contains only rules of the form $\mathsf{a} \leftarrow {\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m$. The conditions set in Definition 1 imply that atomic programs are unary ones and unary programs are binary ones. Consequently, the respective classes of normal programs (denoted by $\mathcal{A}$, $\mathcal{U}$, and $\mathcal{B}$) are ordered by inclusion as follows: $\mathcal{A} \subset \mathcal{U} \subset \mathcal{B} \subset \mathcal{P}$. The last includes also *non-binary* normal programs having at least one rule r with $|\mathrm{B}^+(r)| > 2$.

The same syntactic restrictions can be applied within the class of *positive* normal programs $\mathcal{P}^+ \subset \mathcal{P}$. For instance, any unary positive program consists of rules of the very simple forms $\mathsf{a} \leftarrow$ and $\mathsf{b} \leftarrow \mathsf{c}$. We let the superscripted symbols $\mathcal{A}^+$, $\mathcal{U}^+$, $\mathcal{B}^+$ denote the respective subclasses of $\mathcal{P}^+$, which are ordered analogously $\mathcal{A}^+ \subset \mathcal{U}^+ \subset \mathcal{B}^+ \subset \mathcal{P}^+$. Moreover, $\mathcal{C}^+ \subset \mathcal{C}$ holds for each class $\mathcal{C}$ among $\mathcal{A}$, $\mathcal{U}$, and $\mathcal{B}$.

2.2. *Semantics*

Normal programs can be given a standard model-theoretic semantics. For now, the *Herbrand base* $\mathrm{Hb}(P)$ of a normal logic program P is defined as the set of atoms that appear in the rules of P, although a slightly more general definition will be introduced

---

on finite programs and Herbrand instances, which means that function symbols are not fully covered.

in Section 3.1. An *interpretation* $I \subseteq \mathrm{Hb}(P)$ of a normal program P determines which atoms a of $\mathrm{Hb}(P)$ are *true* ($\mathsf{a} \in I$) and which atoms are *false* ($\mathsf{a} \in \mathrm{Hb}(P) - I$). A negative default literal $\sim\mathsf{a}$ is given a classical interpretation at this point: $I \models \sim\mathsf{a} \iff I \not\models \mathsf{a}$. Given a set of literals L, we define $I \models L \iff I \models l$ for every $l \in L$. The interpretation of rules is similar to that of classical implications: $I \models r$ holds for a rule r iff $\iff I \models \mathrm{B}(r)$ implies $I \models \mathrm{H}(r)$. Finally, an interpretation I is a (classical) model of P ($I \models P$) iff $I \models r$ for every $r \in P$. However, the classical semantics is not sufficient to capture the intended meaning of default literals and we have to distinguish models that are minimal in the following sense.

DEFINITION 2. — *A model $M \models P$ is a minimal model of P if and only if there is no model $M' \models P$ such that $M' \subset M$.*

In particular, every positive program P is guaranteed to possess a unique minimal model of which equals to the intersection of all models of P [LLO 87]. We let $\mathrm{LM}(P)$ stand for this particular model, i.e. the *least model* of P. The least model semantics is inherently *monotonic*: if $P \subseteq P'$ holds for two positive programs, then $\mathrm{LM}(P) \subseteq \mathrm{LM}(P')$. Moreover, the least model $\mathrm{LM}(P)$ can be constructed iteratively as follows; see [LLO 87] for a complete treatment. Define an operator T_P on sets of atoms $A \subseteq \mathrm{Hb}(P)$ by setting $\mathrm{T}_P(A) = \{\mathrm{H}(r) \mid r \in P \text{ and } \mathrm{B}^+(r) \subseteq A\}$. The iteration sequence of the operator T_P is defined inductively: (i) $\mathrm{T}_P \uparrow 0 = \emptyset$, (ii) $\mathrm{T}_P \uparrow i = \mathrm{T}_P(\mathrm{T}_P \uparrow i-1)$ for $i > 0$, and (iii) $\mathrm{T}_P \uparrow \omega = \bigcup_{i<\omega} \mathrm{T}_P \uparrow i$. It follows that $\mathrm{LM}(P) = \mathrm{T}_P \uparrow \omega = \mathrm{lfp}(\mathrm{T}_P)$ which is reached in a finite number of steps if P is finite. Given a positive program P and an atom $\mathsf{a} \in \mathrm{LM}(P)$, we define the *level number* $\mathrm{lev}(\mathsf{a})$ of a as the least natural number i such that $\mathsf{a} \in \mathrm{T}_P \uparrow i$.

Gelfond and Lifschitz [GEL 88] propose a way to apply the least model semantics to an arbitrary normal program P. Given an interpretation $M \subseteq \mathrm{Hb}(P)$, i.e. a model candidate, their idea is to reduce P to a positive program

$$P^M = \{r^+ \mid r \in P \text{ and } M \cap \mathrm{B}^-(r) = \emptyset\}.^2 \tag{2}$$

In this way, the negative default literals appearing in the bodies of the rules of P are simultaneously interpreted with respect to M. Since the reduct P^M is a positive program, it has a natural semantics determined by the least model $\mathrm{LM}(P^M)$. Equating this model with the model candidate M used to reduce P leads to the idea of *stability*.

DEFINITION 3 ([GEL 88]). — *An interpretation $M \subseteq \mathrm{Hb}(P)$ of a normal logic program P is a stable model of $P \iff M = \mathrm{LM}(P^M)$.*

In general, a normal logic program need not have a unique stable model (see P_1 in Example 4 below) nor a stable models at all (see P_2 in Example 4). The minimality of stable models is demonstrated by P_3 in Example 4: $M' = \{\mathsf{a}, \mathsf{b}\}$ is not stable. In contrast to the least models of positive programs, stable models may change in a *non-monotonic* way which implies that conclusions may be retracted under stable model

2. The conditions of Definition 1 have been designed to be compatible with Gelfond-Lifschitz reduction: If P belongs to a class $\mathcal{C} \in \{\mathcal{A}, \mathcal{U}, \mathcal{B}, \mathcal{P}\}$ and $M \subseteq \mathrm{Hb}(P)$, then $P^M \in \mathcal{C}^+$.

semantics. This is demonstrated below using programs Q_1 and Q_2 in Example 4. Normal logic programs are well-suited for a variety of knowledge representation and reasoning tasks. The reader is referred e.g. to [MAR 99, NIE 99] for further examples on using normal logic programs for such tasks in practice.

EXAMPLE 4. — (i) Program $P_1 = \{\mathsf{a} \leftarrow \sim\mathsf{b};\ \ \mathsf{b} \leftarrow \sim\mathsf{a}\}$[3] has two stable models $M_1 = \{\mathsf{a}\}$ and $M_2 = \{\mathsf{b}\}$. (ii) Program $P_2 = \{\mathsf{a} \leftarrow \sim\mathsf{a}\}$ has no stable models. (iii) Program $P_3 = \{\mathsf{a} \leftarrow \mathsf{b};\ \ \mathsf{b} \leftarrow \mathsf{a}\}$ has a unique stable model $M = \emptyset$. (iv) Programs $Q_1 = \{\mathsf{a} \leftarrow \sim\mathsf{b}\}$ and $Q_2 = Q_1 \cup \{\mathsf{b} \leftarrow\}$ have unique stable models $N_1 = \{\mathsf{a}\}$ and $N_2 = \{\mathsf{b}\}$, respectively, but $N_1 \not\subseteq N_2$. □

We emphasize that the semantics of the logic programs in the classes introduced so far is determined by the stable/least model semantics. Nevertheless, we would like to remind the reader about its predecessor, namely the one based on *supported models* [APT 88]. A model $M \subseteq \mathrm{Hb}(P)$ of P is supported by P if and only if for each atom a true in M there is a rule $r \in P$ such that $\mathrm{H}(r) = \mathsf{a}$ and $M \models \mathrm{B}(r)$. In contrast to [APT 88], our definition of T_P covers only positive programs and we have to use P^M to characterize supported models in terms of T_P.

PROPOSITION 5. — *An interpretation $M \subseteq \mathrm{Hb}(P)$ is a supported model of P* $\iff$ $M = \mathrm{T}_{P^M}(M)$.

PROOF. — Now $M \models P \iff M \models P^M \iff \mathrm{T}_{P^M}(M) \subseteq M$ [APT 88, Lemma 2]. On the other hand, the interpretation M is supported by $P \iff M$ is supported by $P^M \iff M \subseteq \mathrm{T}_{P^M}(M)$ [APT 88, Lemma 3]. ■

Thus every stable (resp. supported) model of P is also a supported (resp. classical) model of P [MAR 92] but not necessarily vice versa. For instance, $M' = \{\mathsf{a}, \mathsf{b}\}$ is a supported model of P_3 (resp. $N' = \{\mathsf{b}\}$ is a classical model of Q_1) in Example 4.

2.3. *Propositional theories*

We define *classical literals* in the standard way using classical negation $\neg$ as the connective. A *clause* C is a finite set of classical literals

$$\{\mathsf{a}_1, \ldots, \mathsf{a}_n, \neg\mathsf{b}_1, \ldots, \neg\mathsf{b}_m\} \tag{3}$$

representing a disjunction of its constituents. A *propositional theory in a conjunctive normal form*[4] is a set of clauses S representing a conjunction of the clauses (3) contained in S. In the sequel, the class of propositional theories is denoted by $\mathcal{PT}$.

Given a set of clauses S, we let $\mathrm{Hb}(S)$ denote its Herbrand base so that interpretations I for S can be defined as subsets of $\mathrm{Hb}(S)$ in analogy to Section 2.2. The satisfaction relation $\models$ is defined in the standard way for clauses (3). Similarly to logic programs, a set of clauses S gives rise to a set of models $\{M \subseteq \mathrm{Hb}(S) \mid M \models S\}$,

3. To avoid confusions, we use semicolons ";" to separate rules in logic programs.
4. Without loss of generality under polynomial and faithful translations; see Section 3.3.

but the essential difference is that all classical models are taken into account. A set of clauses S is *satisfiable* if it has at least one model, and *unsatisfiable* otherwise.

3. Translation functions

The author has analyzed the expressive powers of *non-monotonic logics* in a systematic fashion [JAN 99b, JAN 03a, JAN 00a] which extends previous results obtained by Imielinski [IMI 87] and Gottlob [GOT 95]. The comparison is based on the existence/non-existence of *polynomial*, *faithful* and *modular* (PFM) translation functions between non-monotonic logics under consideration. As a result, the expressive power hierarchy (EPH) of non-monotonic logics [JAN 03a] was gradually established. In this section, we propose an analogous framework to compare the expressive powers of classes $\mathcal{C}$ of logic programs. We begin by making certain general assumptions about (classes of) logic programs in Section 3.1. Based on these assumptions we are ready to define the notion of *visible equivalence* in Section 3.2 and to introduce PFM translation functions between classes of logic programs in Section 3.3. The resulting classification method for expressiveness analysis is summarized in Section 3.4.

3.1. *General assumptions about (classes of) logic programs*

At this level of abstraction, logic programs P are understood syntactically as sets of expressions built of propositional atoms. This is to cover also other formalisms in addition to those introduced in Section 2. There we defined the Herbrand base $\mathrm{Hb}(P)$ as the set of atoms that effectively *appear* in P. Basically, we would like to apply the same principle at this level abstraction, but sometimes there is a need to extend $\mathrm{Hb}(P)$ with certain atoms that do not appear in P. This kind of a setting may arise e.g. when a particular logic program P is being optimized. Suppose that an atom $\mathsf{a} \in \mathrm{Hb}(P)$ is recognized useless in the program P and all of its occurrences (and possibly the rules involved) are removed from P. Thus $\mathsf{a} \notin \mathrm{Hb}(P')$ holds for the resulting program P'. Let us mention $P = \{\mathsf{a} \leftarrow \mathsf{a}\}$ and $P' = \emptyset$ as concrete examples of such programs, whose least models coincide. The fact that a is forgotten in this way impedes the comparison of P and P' in a sense (to be made precise in Section 3.2), since their Herbrand bases become different according to our preliminary definition. For this reason, we propose a revised definition: $\mathrm{Hb}(P)$ is any *fixed* set of atoms containing all atoms that actually appear in the rules of P. Thus $\mathrm{Hb}(P)$ acts as the *symbol table* of P which also contributes to the length of the program; viewed now as a pair $\langle P, \mathrm{Hb}(P)\rangle$ rather than just a set of rules. For instance, the pair $\langle \emptyset, \{\mathsf{a}\}\rangle$ properly represents the program P' discussed above.

However, there is a further aspect of atoms that affects the way we treat Herbrand bases, namely the *visibility* of atoms. It is typical in answer set programming that only certain atoms appearing in a program are relevant for representing the solutions of the problem being solved. Others act as auxiliary concepts that can be usually *hidden* from the user altogether. In fact, the choice of such concepts may vary in other programs

written for the same problem. As a side effect, the models assigned to programs may differ already on the basis of auxiliary atoms although they capture the same set of solutions. Rather than introducing a hiding mechanism in the rule language, we let the programmer[5] decide the visible part of $\mathrm{Hb}(P)$, i.e. $\mathrm{Hb}_{\mathrm{v}}(P) \subseteq \mathrm{Hb}(P)$ which determines the set of hidden atoms $\mathrm{Hb}_{\mathrm{h}}(P) = \mathrm{Hb}(P) - \mathrm{Hb}_{\mathrm{v}}(P)$. The ideas presented so far are combined in the following definition.

DEFINITION 6. — *A logic program is a triple* $\langle P, \mathrm{Hb}_{\mathrm{v}}(P), \mathrm{Hb}_{\mathrm{h}}(P)\rangle$ *where*

1) P *is a set of syntactic expressions (e.g. rules) built of propositional atoms,*

2) $\mathrm{Hb}_{\mathrm{v}}(P)$ *and* $\mathrm{Hb}_{\mathrm{h}}(P)$ *are disjoint sets of atoms and determine the visible and hidden Herbrand bases of the program, respectively, and*

3) all atoms occurring in P *are contained in* $\mathrm{Hb}(P) = \mathrm{Hb}_{\mathrm{v}}(P) \cup \mathrm{Hb}_{\mathrm{h}}(P)$.

Finally, we define $\mathrm{Hb}_{\mathrm{a}}(P)$ *as the set of atoms* $\mathsf{a} \in \mathrm{Hb}(P)$ *not occurring in* P.

Note that the atoms of $\mathrm{Hb}_{\mathrm{a}}(P)$ can be viewed as *additional* atoms that just extend $\mathrm{Hb}(P)$. By a slight abuse of notation, we often use P rather than the whole triple when referring to a program $\langle P, \mathrm{Hb}_{\mathrm{v}}(P), \mathrm{Hb}_{\mathrm{h}}(P)\rangle$. To ease the treatment of programs in the sequel, we make some default assumptions regarding the sets $\mathrm{Hb}(P)$ and $\mathrm{Hb}_{\mathrm{v}}(P)$. Unless otherwise stated, we assume that $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}(P)$, $\mathrm{Hb}_{\mathrm{h}}(P) = \emptyset$, and $\mathrm{Hb}_{\mathrm{a}}(P) = \emptyset$, i.e. $\mathrm{Hb}(P)$ contains only atoms that actually appear in P.

EXAMPLE 7. — Given $P = \{\mathsf{a} \leftarrow {\sim}\mathsf{b}\}$, the default interpretation is that $\mathrm{Hb}(P) = \{\mathsf{a}, \mathsf{b}\}$, $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}(P) = \{\mathsf{a}, \mathsf{b}\}$, and $\mathrm{Hb}_{\mathrm{h}}(P) = \emptyset$. To make an exception, we have to add explicitly that $\mathrm{Hb}_{\mathrm{v}}(P) = \{\mathsf{a}, \mathsf{c}\}$ and $\mathrm{Hb}_{\mathrm{h}}(P) = \{\mathsf{b}\}$, for example. □

Having now settled our concerns regarding the Herbrand bases of programs, we make some general assumptions about classes of logic programs. These will be needed when the requirements for translation functions are formulated in Sections 3.2 and 3.3.

DEFINITION 8. — *Any class* $\mathcal{C}$ *of logic programs must satisfy the following.*

A1. Every $P \in \mathcal{C}$ *and its Herbrand bases* $\mathrm{Hb}_{\mathrm{v}}(P)$ *and* $\mathrm{Hb}_{\mathrm{h}}(P)$ *are* ***finite****.*

A2. The class $\mathcal{C}$ *has a* ***semantic operator*** $\mathrm{Sem}_{\mathcal{C}}$ *which maps a program* $P \in \mathcal{C}$ *to a set of interpretations* $\mathrm{Sem}_{\mathcal{C}}(P) \subseteq \mathbf{2}^{\mathrm{Hb}(P)}$ *which determines the semantics of* P.

The first assumption reflects the fact that we take propositional logic programs as potential inputs to translator programs as well as solvers that compute models for them. Hence we must be able to encode any program under consideration as a finite string of symbols. This excludes the possibility of instantiating a logic program involving variables and function symbols to a fully propositional one. By the second assumption, the semantics of each program P in a class $\mathcal{C}$ is determined by a set of *total* [6] interpretations $\mathrm{Sem}_{\mathcal{C}}(P)$. Given an interpretation $I \in \mathrm{Sem}_{\mathcal{C}}(P)$, each atom

5. E.g., the front-end of the SMODELS system [SIM 02] enables visibility control in terms of *hide* and *show* statements.

6. It is quite possible to generalize A2 to cover *partial models* like the well-wounded model [VAN 91], but such models are not addressed in this article.

$\mathsf{a} \in \mathrm{Hb}(P)$ is assigned to either true or false as done in Sections 2.2 and 2.3. Hence we assume a two-valued semantics for each class of logic programs in this article.

It is easy to see that the finite fragments of the classes of logic programs $\mathcal{C}$ introduced in Section 2.1 satisfy these assumptions. In the sequel, we identify these classes with their finite fragments. The semantic operator is given below in (4). Note that the stable semantics coincides with the least model semantics in the case of positive programs. Due to flexibility of Definitions 6 and 8 it is possible to view the class of finite propositional theories $\mathcal{PT}$ as a class of logic programs given the respective semantic operator in (5). This enables the comparison of these classes in sections to come.

$$\mathrm{Sem}_{\mathcal{C}}(P) = \mathrm{SM}(P) = \{M \subseteq \mathrm{Hb}(P) \mid M = \mathrm{LM}(P^M)\} \tag{4}$$

$$\mathrm{Sem}_{\mathcal{PT}}(S) = \mathrm{CM}(S) = \{M \subseteq \mathrm{Hb}(S) \mid M \models S\} \tag{5}$$

3.2. *Visible equivalence*

Having defined the semantics of logic programs on an abstract level, the next issue is to define the conditions on which two representatives P and Q of a given class of logic programs $\mathcal{C}$ can be considered to be equivalent. It is natural that the answer to this question goes back to semantics. A straightforward notion of equivalence is obtained by equating $\mathrm{Sem}_{\mathcal{C}}(P)$ and $\mathrm{Sem}_{\mathcal{C}}(Q)$. This corresponds to the basic notion of equivalence proposed for normal programs under stable model semantics, but stronger notions have also been proposed. For instance, Lifschitz et al. [LIF 01] consider P and Q *strongly equivalent* given that $\mathrm{Sem}_{\mathcal{C}}(P \cup R) = \mathrm{Sem}_{\mathcal{C}}(Q \cup R)$ for all other programs R. Consequently, the strong equivalence of P and Q implies that P and Q can be freely used as substitutes of each other. Although such a notion seems attractive at the first glance, a drawback is that it is rather restrictive — allowing only relatively straightforward semantics-preserving modifications to rules [EIT 04].

Both approaches share another problem: the models in $\mathrm{Sem}_{\mathcal{C}}(P)$ and $\mathrm{Sem}_{\mathcal{C}}(Q)$ have to be identical subsets of $\mathrm{Hb}(P)$ and $\mathrm{Hb}(Q)$, respectively. Therefore, we propose a notion of equivalence which tries to take the interfaces of logic programs properly into account. The idea is that atoms in $\mathrm{Hb}_{\mathrm{h}}(P)$ and $\mathrm{Hb}_{\mathrm{h}}(Q)$ are considered as local to P and Q and negligible as far as the equivalence of the programs is concerned.

DEFINITION 9. — *Logic programs $P \in \mathcal{C}$ and $Q \in \mathcal{C}'$ 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 $f : \mathrm{Sem}_{\mathcal{C}}(P) \rightarrow \mathrm{Sem}_{\mathcal{C}'}(Q)$ such that for every $M \in \mathrm{Sem}_{\mathcal{C}}(P)$,*

$$M \cap \mathrm{Hb}_{\mathrm{v}}(P) = f(M) \cap \mathrm{Hb}_{\mathrm{v}}(Q). \tag{6}$$

Note that this notion of equivalence can be applied both within a single class of logic programs, and between different classes, which may be syntactically and/or se-

mantically different. This is a very important aspect, as we intend to study the interrelationships of such classes of programs in the sequel.

EXAMPLE 10. — Consider a program $P = \{\mathsf{a} \leftarrow \sim\mathsf{b};\ \mathsf{b} \leftarrow \sim\mathsf{a};\ \mathsf{c} \leftarrow \mathsf{a};\ \mathsf{c} \leftarrow \sim\mathsf{a}\}$ with $\mathrm{Hb}_{\mathrm{v}}(P) = \{\mathsf{a}, \mathsf{c}\}$ and stable models $M_1 = \{\mathsf{a}, \mathsf{c}\}$ and $M_2 = \{\mathsf{b}, \mathsf{c}\}$. Thus $\mathrm{Hb}_{\mathrm{h}}(P) = \{\mathsf{b}\}$ remains hidden when we compare P with a set of clauses $S = \{\{\mathsf{a}, \mathsf{d}\},\ \{\neg\mathsf{a}, \neg\mathsf{d}\},\ \{\mathsf{a}, \mathsf{c}\},\ \{\neg\mathsf{a}, \mathsf{c}\}\}$ possessing exactly two classical models $N_1 = \{\mathsf{a}, \mathsf{c}\}$ and $N_2 = \{\mathsf{d}, \mathsf{c}\}$, as $\mathrm{Hb}(S) = \{\mathsf{a}, \mathsf{c}, \mathsf{d}\}$. We can hide d by setting $\mathrm{Hb}_{\mathrm{v}}(S) = \{\mathsf{a}, \mathsf{c}\}$. Then $P \equiv_{\mathrm{v}} S$ holds, as $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(S)$ and there is a bijection from $f : \mathrm{SM}(P) \to \mathrm{CM}(S)$ that (i) maps M_1 to N_1 so that $M_1 \cap \mathrm{Hb}_{\mathrm{v}}(P) = \{\mathsf{a}, \mathsf{c}\} = N_1 \cap \mathrm{Hb}_{\mathrm{v}}(Q)$, and (ii) maps M_2 to N_2 so that $M_2 \cap \mathrm{Hb}_{\mathrm{v}}(P) = \{\mathsf{c}\} = N_2 \cap \mathrm{Hb}_{\mathrm{v}}(Q)$. □

PROPOSITION 11. — $\equiv_{\mathrm{v}}$ *is an equivalence relation among all programs.*

PROOF. — The reflexivity of $\equiv_{\mathrm{v}}$ follows essentially by the identity mapping $i : \mathrm{Sem}_{\mathcal{C}}(P) \to \mathrm{Sem}_{\mathcal{C}}(P)$ for any P from any $\mathcal{C}$. The symmetry of $\equiv_{\mathrm{v}}$ is also easily obtained. Given $P \equiv_{\mathrm{v}} Q$ for any programs P and Q from any classes $\mathcal{C}$ and $\mathcal{C}'$, the existence of an inverse for a bijection $f : \mathrm{Sem}_{\mathcal{C}}(P) \to \mathrm{Sem}_{\mathcal{C}'}(Q)$ is guaranteed. For the transitivity of $\equiv_{\mathrm{v}}$, let f_1 and f_2 be the respective bijections involved in $P \equiv_{\mathrm{v}} Q$ and $Q \equiv_{\mathrm{v}} R$ where $P \in \mathcal{C}_1$, $Q \in \mathcal{C}_2$, and $R \in \mathcal{C}_3$. It is clear that $f_1 \circ f_2$ is also a bijection, and we have for all $M \in \mathrm{Sem}_{\mathcal{C}_1}(P)$ that $M \cap \mathrm{Hb}_{\mathrm{v}}(P) = f_1(M) \cap \mathrm{Hb}_{\mathrm{v}}(Q) = f_2(f_1(M)) \cap \mathrm{Hb}_{\mathrm{v}}(R) = (f_1 \circ f_2)(M) \cap \mathrm{Hb}_{\mathrm{v}}(R)$. Moreover, $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$ and $\mathrm{Hb}_{\mathrm{v}}(Q) = \mathrm{Hb}_{\mathrm{v}}(R)$ imply $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(R)$. Thus $P \equiv_{\mathrm{v}} R$. ■

It is worthwhile to do some comparisons. By setting $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}(P)$ and $\mathrm{Hb}_{\mathrm{v}}(Q) = \mathrm{Hb}(Q)$, the relation $\equiv_{\mathrm{v}}$ becomes very close to the notion of weak equivalence discussed in the beginning of this section, if interpreted with respect to the class of normal programs under the stable model semantics. The only difference is the implied additional requirement that $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$ in order to $P \equiv Q$ to hold. By the approach taken in Section 3.1 this requirement becomes of little account: Herbrand bases are always extendible to meet $\mathrm{Hb}(P) = \mathrm{Hb}(Q)$. Actually, we can state the same about $\equiv_{\mathrm{v}}$ using a generalized notion of weak equivalence: $P \equiv Q$ is defined to hold for $P \in \mathcal{C}$ and $Q \in \mathcal{C}' \iff \mathrm{Sem}_{\mathcal{C}}(P) = \mathrm{Sem}_{\mathcal{C}'}(Q)$. It is clear that $\equiv$ is also an equivalence relation among all programs.

PROPOSITION 12 ([JAN 03B]). — *If P and Q are logic programs having equal Herbrand bases $\mathrm{Hb}_{\mathrm{v}}(P)$, $\mathrm{Hb}(P)$, $\mathrm{Hb}_{\mathrm{v}}(Q)$, and $\mathrm{Hb}(Q)$, then $P \equiv Q \iff P \equiv_{\mathrm{v}} Q$.*

As regards the equivalence of logic programs, an alternative approach is to use their *skeptical* and *brave* consequences as a criterion. A propositional formula ϕ based on $\mathrm{Hb}_{\mathrm{v}}(P)$ is a skeptical (resp. brave) consequence of P iff $M \models \phi$ for all (resp. for some) $M \in \mathrm{SM}(P)$. For the sake of comparison, we define P and Q to be equivalent in the skeptical (resp. brave) sense, denoted $P \equiv_{\mathrm{vs}} Q$ (resp. $P \equiv_{\mathrm{vb}} Q$) iff $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$ and the skeptical (resp. brave) consequences of P and Q are the same. Yet weaker notion of equivalence is obtained by using *consistency* as the only criteria: $P \equiv_{\mathrm{vc}} Q$ iff $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$ and $\mathrm{SM}(P) = \emptyset \iff \mathrm{SM}(Q) = \emptyset$. These alternative notions of equivalence relate with visible equivalence as follows.

PROPOSITION 13. — *Let P and Q be two logic programs. Then $P \equiv_{\mathrm{v}} Q$ implies both $P \equiv_{\mathrm{vs}} Q$ and $P \equiv_{\mathrm{vb}} Q$ which, in turn, separately imply $P \equiv_{\mathrm{vc}} Q$.*

PROOF. — Suppose $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$ since these implications hold otherwise. (i) If $P \not\equiv_{\mathrm{vc}} Q$, then we may assume $\mathrm{SM}(P) = \emptyset$ and $\mathrm{SM}(Q) \neq \emptyset$ without loss of generality. It follows that $\bot$ is a skeptical consequence of P but not that of Q, i.e. $P \not\equiv_{\mathrm{vs}} Q$. On the other hand, $\top$ is a brave consequence of Q but not that of P which indicates that $P \not\equiv_{\mathrm{vb}} Q$. (ii) If $P \not\equiv_{\mathrm{vs}} Q$, we may assume without loss of generality that Q has a skeptical consequence ϕ based on $\mathrm{Hb}_{\mathrm{v}}(Q) = \mathrm{Hb}_{\mathrm{v}}(P)$ which is not that of P. Then there is $M \in \mathrm{SM}(P)$ such that $M \not\models \phi$. Assuming $P \equiv_{\mathrm{v}} Q$, there is a bijection f so that $N = f(M) \in \mathrm{SM}(Q)$ coincides with with M up to $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$. It follows that $N \not\models \phi$. A contradiction, as ϕ is a skeptical consequence of Q. Hence $P \not\equiv_{\mathrm{v}} Q$. (iii) Analogously, if $P \not\equiv_{\mathrm{vb}} Q$, we may assume a brave consequence ϕ of P which is based on $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q)$ and which is not a brave consequence of Q. It follows that $M \models \phi$ for some $M \in \mathrm{SM}(P)$. Let us then assume $P \equiv_{\mathrm{v}} Q$ and let f be the respective bijection for stable models. Then $N \models \phi$ holds for $N = f(M) \in \mathrm{Hb}(Q)$, a contradiction. Thus $P \not\equiv_{\mathrm{v}} Q$. ■

The converse implications do not hold in general. For $P = \{\mathsf{a} \leftarrow {\sim}\mathsf{b}\}$ and $Q = \emptyset$ with $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q) = \{\mathsf{a}\}$, we have $\mathrm{SM}(P) = \{\{\mathsf{a}\}\}$ and $\mathrm{SM}(Q) = \{\emptyset\}$ so that $P \equiv_{\mathrm{vc}} Q$. However, a is a skeptical/brave conclusion of P but not that of Q. Thus $P \not\equiv_{\mathrm{vs}} Q$ and $P \not\equiv_{\mathrm{vb}} Q$. On the other hand, let us consider $P = \{\mathsf{a} \leftarrow \mathsf{b}; \mathsf{a} \leftarrow \mathsf{c}; \mathsf{b} \leftarrow {\sim}\mathsf{c}; \mathsf{c} \leftarrow {\sim}\mathsf{b}\}$ and $Q = \{\mathsf{a} \leftarrow\}$ with $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(Q) = \{\mathsf{a}\}$. Now $P \equiv_{\mathrm{vs}} Q$ and $P \equiv_{\mathrm{vb}} Q$ as $\mathrm{SM}(P) = \{\{\mathsf{a}, \mathsf{b}\}, \{\mathsf{a}, \mathsf{c}\}\}$ and $\mathrm{SM}(Q) = \{\{\mathsf{a}\}\}$. But a bijective correspondence between $\mathrm{SM}(P)$ and $\mathrm{SM}(Q)$ is impossible. Thus $P \not\equiv_{\mathrm{v}} Q$.

3.3. *Requirements for translation functions*

We are now ready to formulate our criteria for a translation function $\mathrm{Tr} : \mathcal{C} \to \mathcal{C}'$ that transforms logic programs P of one class $\mathcal{C}$ into logic programs $\mathrm{Tr}(P)$ of another class $\mathcal{C}'$. In many cases of interest, the latter class is a subclass or a superclass of $\mathcal{C}$, but it makes also sense to perform translations between classes that are incomparable in this respect (such as $\mathcal{P}$ and $\mathcal{PT}$ introduced so far). It is assumed below that both the source class $\mathcal{C}$ and the target class $\mathcal{C}'$ satisfy assumptions listed in Definition 8.

DEFINITION 14. — *A translation function $\mathrm{Tr} : \mathcal{C} \to \mathcal{C}'$ is **polynomial** if and only if for all programs $P \in \mathcal{C}$, the time required to compute the translation $\mathrm{Tr}(P) \in \mathcal{C}'$ is polynomial in $||P||$, i.e. the length of P in symbols.*

Note that the length of the translation $||\mathrm{Tr}(P)||$ is also polynomial in $||P||$ if Tr is polynomial. In many cases, even linear time translation functions can be devised for particular classes of logic programs. Such transformations are highly desirable to allow efficient transformation of knowledge from one representation to another.

DEFINITION 15. — *A translation function $\mathrm{Tr} : \mathcal{C} \to \mathcal{C}'$ is **faithful** if and only if for all $P \in \mathcal{C}$, $P \equiv_{\mathrm{v}} \mathrm{Tr}(P)$.*

Here we emphasize that $P \equiv_{\mathrm{v}} \mathrm{Tr}(P)$ implies $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}(P))$ by the definition of $\equiv_{\mathrm{v}}$. Thus a faithful translation function Tr may introduce new atoms, which have to remain invisible, or forget old invisible atoms. Moreover, if we insist on polynomiality, then the number of new atoms gets limited, too. The possibility of introducing new atoms is a crucial option for translation functions to be presented in the sequel. This is because new atoms serve as shorthands for more complex logical expressions that save space and enable translation functions between certain classes.

The existence of a bijection f between $\mathrm{Sem}_{\mathcal{C}}(P)$ and $\mathrm{Sem}_{\mathcal{C}'}(\mathrm{Tr}(P))$ ensures that the semantics of P is captured by $\mathrm{Tr}(P)$ to a sufficient degree as regards ASP: there is a bijective correspondence of models and the models coincide up to $\mathrm{Hb}_{\mathrm{v}}(P)$. Recall that in ASP the models in $\mathrm{Sem}_{\mathcal{C}}(P)$ correspond to the solutions of the problem being solved, and such a tight relationship is necessary in order to preserve the *number* of solutions. This is an important aspect often neglected in literature. The number of solutions would not be preserved if the notion of faithfulness were weakened by replacing $\equiv_{\mathrm{v}}$ in Definition 15 with any of the weaker equivalence relations $\equiv_{\mathrm{vs}}$, $\equiv_{\mathrm{vb}}$, and $\equiv_{\mathrm{vc}}$ considered in Proposition 13. On the other hand, if a particular translation function Tr is proved faithful in the sense of Definition 15, then also $P \equiv_{\mathrm{vs}} \mathrm{Tr}(P)$, $P \equiv_{\mathrm{vb}} \mathrm{Tr}(P)$, and $P \equiv_{\mathrm{vc}} \mathrm{Tr}(P)$ hold for all $P \in \mathcal{C}$, i.e. Tr preserves skeptical and brave conclusions as well as consistency within the language determined by $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}(P))$. Nevertheless, any intranslatability results may be affected by such weakenings. E.g., if a faithful translation is proved non-existent using $\equiv_{\mathrm{v}}$, the proof may not be valid if weaker notions of faithfulness are taken into consideration.

The third requirement for translation functions, namely *modularity*, is based on the idea of combining *disjoint* program modules together. The module conditions below make precise which combinations of programs are considered appropriate.

DEFINITION 16. — *Logic programs $P \in \mathcal{C}$ and $Q \in \mathcal{C}$ satisfy* ***module conditions*** *if and only if*

M1. $P \cap Q = \emptyset$,

M2. $\mathrm{Hb}_{\mathrm{a}}(P) \cap \mathrm{Hb}_{\mathrm{a}}(Q) = \emptyset$,[7]

M3. $\mathrm{Hb}_{\mathrm{h}}(P) \cap \mathrm{Hb}(Q) = \emptyset$, *and*

M4. $\mathrm{Hb}(P) \cap \mathrm{Hb}_{\mathrm{h}}(Q) = \emptyset$.

DEFINITION 17. — *A translation function $\mathrm{Tr} : \mathcal{C} \to \mathcal{C}'$ is* ***modular*** *if and only if for all programs $P \in \mathcal{C}$ and $Q \in \mathcal{C}$ satisfying module conditions M1–M4,*

$$\mathrm{Tr}(P \cup Q) = \mathrm{Tr}(P) \cup \mathrm{Tr}(Q) \tag{7}$$

and the translations $\mathrm{Tr}(P)$ and $\mathrm{Tr}(Q)$ satisfy module conditions M1–M4.

The aim of the modularity condition is to enforce the locality of Tr with respect to subprograms P and Q which can be viewed as program modules that interact through

7. To see why M2 is analogous to M1, note that for a normal program P, an atom $\mathsf{a} \in \mathrm{Hb}_{\mathrm{a}}(P)$ is parallel to a *useless* (tautological) rule $\mathsf{a} \leftarrow \mathsf{a} \in P$ such that a does not occur elsewhere in the program. Such rules do not affect stable models but may create new classical models.

visible atoms only. By (7), the modules P and Q have to be separately translatable and the translation $\mathrm{Tr}(P \cup Q)$ is obtained as the union of the translations of the modules. In addition, a modular translation function is supposed to preserve module conditions M1–M4, i.e. the respective translations $\mathrm{Tr}(P)$ and $\mathrm{Tr}(Q)$ are supposed to remain disjoint and share only visible atoms. The modularity condition becomes void when programs share rules or hidden atoms.

PROPOSITION 18 ([JAN 03B]). — *If* $\mathrm{Tr}_1 : \mathcal{C}_1 \to \mathcal{C}_2$ *and* $\mathrm{Tr}_2 : \mathcal{C}_2 \to \mathcal{C}_3$ *are two polynomial, faithful, or modular translation functions, then their composition* $\mathrm{Tr}_1 \circ \mathrm{Tr}_2 : \mathcal{C}_1 \to \mathcal{C}_3$ *is also polynomial, faithful, or modular, respectively.*

3.4. *Classification method*

The criteria collected in Definitions 14–17 lead to a method for comparing classes of logic programs on the basis of their expressive power. We say that a translation function $\mathrm{Tr} : \mathcal{C} \to \mathcal{C}'$ is PFM if and only if it is polynomial, faithful, and modular simultaneously. If there exists such a translation function Tr, we write $\mathcal{C} \leq_{\mathrm{PFM}} \mathcal{C}'$ to denote that the class $\mathcal{C}'$ *is at least as expressive as* the class $\mathcal{C}$. This is simply because the essentials of any program $P \in \mathcal{C}$ can be captured using the translation $\mathrm{Tr}(P) \in \mathcal{C}'$. In certain cases, we are able to construct a counter-example which shows that a PFM translation function is impossible, denoted by $\mathcal{C} \not\leq_{\mathrm{PFM}} \mathcal{C}'$. The base relations $\leq_{\mathrm{PFM}}$ and $\not\leq_{\mathrm{PFM}}$ among classes of logic programs form the cornerstones of the classification method – giving rise to relations given in Table 1.

Table 1. *Relations used by the classification method*

Relation	Definition	Explanation
$\mathcal{C} <_{\mathrm{PFM}} \mathcal{C}'$	$\mathcal{C} \leq_{\mathrm{PFM}} \mathcal{C}'$ and $\mathcal{C}' \not\leq_{\mathrm{PFM}} \mathcal{C}$	$\mathcal{C}$ is *strictly less expressive* than $\mathcal{C}'$
$\mathcal{C} =_{\mathrm{PFM}} \mathcal{C}'$	$\mathcal{C} \leq_{\mathrm{PFM}} \mathcal{C}'$ and $\mathcal{C}' \leq_{\mathrm{PFM}} \mathcal{C}$	$\mathcal{C}$ and $\mathcal{C}'$ are *equally expressive*
$\mathcal{C} \neq_{\mathrm{PFM}} \mathcal{C}'$	$\mathcal{C} \not\leq_{\mathrm{PFM}} \mathcal{C}'$ and $\mathcal{C}' \not\leq_{\mathrm{PFM}} \mathcal{C}$	$\mathcal{C}$ and $\mathcal{C}'$ are *incomparable*

It is sometimes convenient to introduce variants of the relations $\leq_{\mathrm{PFM}}$ and $\not\leq_{\mathrm{PFM}}$ which are obtained by dropping some of the three requirements and the corresponding letter(s) in the notation. For instance, if we establish $\mathcal{C} \not\leq_{\mathrm{FM}} \mathcal{C}'$ for certain classes $\mathcal{C}$ and $\mathcal{C}'$ of logic programs, then $\mathcal{C} \not\leq_{\mathrm{PFM}} \mathcal{C}'$ follows immediately. Also, *non-modular* translation functions will be addressed in this article and the resulting relationships involve $\leq_{\mathrm{PF}}$ rather than $\leq_{\mathrm{PFM}}$. In certain cases, it is easy to establish relationships regarding $\leq_{\mathrm{PFM}}$. By the following, we address a frequently appearing case where the syntax is generalized while the semantics is kept compatible with the original one.

PROPOSITION 19 ([JAN 03B]). — *If* $\mathcal{C}$ *and* $\mathcal{C}'$ *are two classes of logic programs such that* $\mathcal{C} \subseteq \mathcal{C}'$ *and* $\mathrm{Sem}_{\mathcal{C}}(P) = \mathrm{Sem}_{\mathcal{C}'}(P)$ *for all* $P \in \mathcal{C}$*, then* $\mathcal{C} \leq_{\mathrm{PFM}} \mathcal{C}'$.

In many cases, we manage to construct faithful translation functions that only add new hidden atoms to programs being translated. The following proposition provides us with a sufficient set of conditions using which faithfulness can be proved in a systematic fashion. The proof [JAN 03b] employs the second condition to establish that Ext is indeed an injective function from $\mathrm{Sem}_{\mathcal{C}}(P)$ to $\mathrm{Sem}_{\mathcal{C}'}(\mathrm{Tr}(P))$ whereas the third condition ensures that Ext is a surjection and thus also a bijection.

PROPOSITION 20. — *A translation function* $\mathrm{Tr} : \mathcal{C} \to \mathcal{C}'$ *is faithful if for every program* $P \in \mathcal{C}$*,*

1) $\mathrm{Hb}(P) \subseteq \mathrm{Hb}(\mathrm{Tr}(P))$ *and* $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}(P)) = \mathrm{Hb}_{\mathrm{v}}(P)$*;*

2) there is an extension operator $\mathrm{Ext} : \mathrm{Sem}_{\mathcal{C}}(P) \to \mathrm{Sem}_{\mathcal{C}'}(\mathrm{Tr}(P))$ *such that* $\forall M \in \mathrm{Sem}_{\mathcal{C}}(P) : M = \mathrm{Ext}(M) \cap \mathrm{Hb}(P)$*; and*

3) if $N \in \mathrm{Sem}_{\mathcal{C}'}(\mathrm{Tr}(P))$*, then* $M = N \cap \mathrm{Hb}(P) \in \mathrm{Sem}_{\mathcal{C}}(P)$ *such that* $N = \mathrm{Ext}(M)$*.*

4. Expressive power analysis

In this section, we compare the expressive powers of the classes of logic programs introduced in Section 2.1 using the classification method presented in Section 3.4. Due to the nature of the syntactic constraints, the key problem is to see whether there are ways to reduce the number of positive body literals in the bodies of rules. The results of this section will indicate that this is possible to some extent, but not in general, i.e. there is no faithful and modular way of removing *all* positive body literals from rules. As a preparation for the expressive power analysis, we distinguish certain properties program modules and positive programs in Section 4.1. Then we analyze the expressiveness of the class of normal programs $\mathcal{P}$ and its subclasses ($\mathcal{A}$, $\mathcal{U}$, and $\mathcal{B}$) in Section 4.2. The stable model semantics in its full generality makes the analysis rather intricate and involved. The analysis of positive programs is much easier and we basically skip it by concluding analogous relationships for the classes of positive programs. Finally, we take classical propositional logic into consideration in Section 4.3 and relate sets of clauses under classical models with the other classes.

4.1. *Some properties of program modules and positive programs*

We prepare the forthcoming expressive power analysis by presenting two useful properties of program modules under the least/stable model semantics. The first property is related with a positive program $P \cup Q$ consisting of two subprograms P and Q so that the module conditions M1–M4 from Definition 16 are satisfied. Here the aim is to provide sufficient conditions for removing either one of the modules by evaluating its effect on the joint least model $\mathrm{LM}(P \cup Q)$ and by replacing it with a compensating atomic program. Formally, we propose a reduction that yields a set of atomic rules.

DEFINITION 21. — *Given a positive normal program $P \in \mathcal{P}^+$ and an interpretation I, the* ***visible net reduction*** *of P is $P_I^{\mathrm{v}} = \{\mathsf{a} \leftarrow \mid \mathsf{a} \in \mathrm{Hb}_{\mathrm{v}}(P) \cap I\}$ so that $\mathrm{Hb}_{\mathrm{v}}(P_I^{\mathrm{v}}) = \mathrm{Hb}(P_I^{\mathrm{v}}) = \mathrm{Hb}_{\mathrm{v}}(P)$ which makes all atoms of P_I^{v} visible.*

The reduced program P_I^{v} overestimates P in a sense, since $\mathsf{a} \leftarrow$ may be included in P_I^{v} even if there is no rule $r \in P$ such that $\mathrm{H}(r) = \mathsf{a}$. However, the reduct P_I^{v} can be formed externally without knowing exactly which rules constitute the program P being reduced. In addition, we assumed that the interpretation I in Definition 21 may contain atoms outside $\mathrm{Hb}(P)$. This setting is easily realized when P is placed as a program module in the context of another program Q. If I is a model for $P \cup Q$, then the set of atoms encoded as atomic rules in P_I^{v} can be understood as the maximum contribution of the rules of P for the atoms that are true in I. In the sequel, we will apply the visible net reduction w.r.t. least models in the following way.

LEMMA 22 ([JAN 03B]). — *Let P and Q be two positive programs satisfying the module conditions M1–M4 and $M = \mathrm{LM}(P \cup Q) \subseteq \mathrm{Hb}(P) \cup \mathrm{Hb}(Q)$. Then $\mathrm{LM}(P_M^{\mathrm{v}} \cup Q) = M \cap (\mathrm{Hb}_{\mathrm{v}}(P) \cup \mathrm{Hb}(Q))$ holds for the visible net reduct P_M^{v}.*

The second property allows us to combine stable models of program modules under certain circumstances to form a stable model for a larger program.

LEMMA 23 ([JAN 03B]). — *Let P and Q be two normal programs satisfying the module conditions. If $M \in \mathrm{SM}(P)$, $N \in \mathrm{SM}(Q)$, and $M \cap \mathrm{Hb}_{\mathrm{v}}(P) \cap \mathrm{Hb}_{\mathrm{v}}(Q) = N \cap \mathrm{Hb}_{\mathrm{v}}(P) \cap \mathrm{Hb}_{\mathrm{v}}(Q)$, then $M \cup N \in \mathrm{SM}(P \cup Q)$ and $(P \cup Q)^{M \cup N} = P^M \cup Q^N$.*

We need a subsidiary result about *unary programs* P: if an atom a is included in $\mathrm{LM}(P)$, then P contains at least one atomic rule which causes the atom a to be inferable by the strictly unary rules of P, i.e. to be included in $\mathrm{LM}(P)$. Note that $\mathrm{LM}(P) = \emptyset$ for any strictly unary program P.

LEMMA 24. — *Let $P = P_1 \cup P_0$ be a unary positive program where P_1 contains the strictly unary rules of P and P_0 contains the atomic rules of P. If $\mathsf{a} \in \mathrm{LM}(P)$, then P_0 contains an atomic rule $\mathsf{b} \leftarrow$ such that $\mathsf{a} \in \mathrm{LM}(P_1 \cup \{\mathsf{b} \leftarrow\})$.*

PROOF. — We use complete induction on the level number $\mathrm{lev}(\mathsf{a}) > 0$ to prove the claim for any atom $\mathsf{a} \in \mathrm{LM}(P_1 \cup P_0) = \mathrm{LM}(P)$.

For the base case, assume that $\mathrm{lev}(\mathsf{a}) = 1$ which implies that $\mathsf{a} \in \mathrm{T}_{P_1 \cup P_0} \uparrow 1 = \mathrm{T}_{P_1 \cup P_0}(\emptyset)$. Thus $\mathsf{a} \leftarrow$ must appear in P_0. It is also clear that $\mathsf{a} \in \mathrm{T}_{P_1 \cup \{\mathsf{a} \leftarrow\}} \uparrow 1 = \mathrm{T}_{P_1 \cup \{\mathsf{a} \leftarrow\}}(\emptyset)$ so that $\mathsf{a} \in \mathrm{LM}(P_1 \cup \{\mathsf{a} \leftarrow\})$.

Then consider the case that $\mathrm{lev}(\mathsf{a}) = i > 1$. Then $\mathsf{a} \in \mathrm{T}_{P_1 \cup P_0} \uparrow i$ and there is a unary rule $\mathsf{a} \leftarrow \mathsf{b} \in P_1$ such that $\mathsf{b} \in \mathrm{T}_{P_1 \cup P_0} \uparrow i - 1$. Since $\mathsf{b} \in \mathrm{LM}(P_1 \cup P_0)$ and $0 < \mathrm{lev}(\mathsf{b}) < \mathrm{lev}(\mathsf{a}) = i$, it follows by the inductive hypothesis that there is an atomic rule $\mathsf{c} \leftarrow$ in P_0 such that $\mathsf{b} \in \mathrm{LM}(P_1 \cup \{\mathsf{c} \leftarrow\})$. This implies $\mathsf{a} \in \mathrm{LM}(P_1 \cup \{\mathsf{c} \leftarrow\})$, because the rule $\mathsf{a} \leftarrow \mathsf{b}$ belongs to P_1. ■

LEMMA 25. — *For positive atomic programs P, $\mathrm{LM}(P) = \{\mathrm{H}(r) \mid r \in P\} = \mathrm{H}(P)$.*

PROOF. — Obviously, we have $\mathrm{LM}(P) = \mathrm{lfp}(\mathrm{T}_P) = \mathrm{T}_P(\emptyset) = \{\mathrm{H}(r) \mid r \in P\}$. ■

4.2. *Expressiveness analysis of normal programs*

In Section 2.1, we identify three subclasses of $\mathcal{P}$ which are obtained by restricting the syntax of the rules whereas the semantics of logic programs in these classes remains unchanged. Thus we obtain the relationships $\mathcal{A} \leq_{\mathrm{PFM}} \mathcal{U} \leq_{\mathrm{PFM}} \mathcal{B} \leq_{\mathrm{PFM}} \mathcal{P}$ directly by Proposition 19, but it remains open whether these relationships are strict or not. We begin with a study of the relationship $\mathcal{B} \leq_{\mathrm{PFM}} \mathcal{P}$. In fact, any *non-binary* rule $\mathsf{a} \leftarrow \mathsf{b}_1, \ldots, \mathsf{b}_n, {\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m$ where $n > 2$ can be rewritten to reduce the number of positive body literals that appear in the rule. One technique is to introduce $n-1$ new atoms $\mathsf{a}^r_1, \ldots, \mathsf{a}^r_{n-1}$ and a set of binary rules $\mathrm{Tr}_{\mathrm{BIN}}(r)$:

$$\begin{array}{l} \mathsf{a} \leftarrow \mathsf{b}_1, \mathsf{a}^r_1, {\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m; \\ \mathsf{a}^r_1 \leftarrow \mathsf{b}_2, \mathsf{a}^r_2; \ \ \ldots; \ \ \mathsf{a}^r_{n-2} \leftarrow \mathsf{b}_{n-1}, \mathsf{a}^r_{n-1}; \\ \mathsf{a}^r_{n-1} \leftarrow \mathsf{b}_n. \end{array} \tag{8}$$

It may be tempting to copy the negative body literals ${\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m$ to every rule in (8), but that would lead to a quadratic translation and we prefer a linear one for the sake of efficiency. The new atoms $\mathsf{a}^r_1, \ldots, \mathsf{a}^r_{n-1}$ carry the identity or r because they are supposed to be *local*[8] to r and hidden in $\mathrm{Tr}_{\mathrm{BIN}}(r)$. This arrangement ensures that the module conditions M3 and M4, i.e. $\mathrm{Hb}_{\mathrm{h}}(\mathrm{Tr}_{\mathrm{BIN}}(r_1)) \cap \mathrm{Hb}(\mathrm{Tr}_{\mathrm{BIN}}(r_2)) = \emptyset$ and $\mathrm{Hb}(\mathrm{Tr}_{\mathrm{BIN}}(r_1)) \cap \mathrm{Hb}_{\mathrm{h}}(\mathrm{Tr}_{\mathrm{BIN}}(r_2)) = \emptyset$, hold for any two non-binary rules r_1 and r_2 sharing no hidden atoms. The translation $\mathrm{Tr}_{\mathrm{BIN}}$ extends for programs as follows.

DEFINITION 26. — *For every $P \in \mathcal{P}$, define* $\mathrm{Tr}_{\mathrm{BIN}}(P) =$

$$\{r \mid r \in P \textit{ and } |\mathrm{B}^+(r)| \leq 2\} \cup \bigcup \{\mathrm{Tr}_{\mathrm{BIN}}(r) \mid r \in P \textit{ and } |\mathrm{B}^+(r)| > 2\}. \tag{9}$$

Moreover, let $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{BIN}}(P)) = \mathrm{Hb}_{\mathrm{v}}(P)$ *and* $\mathrm{Hb}_{\mathrm{a}}(\mathrm{Tr}_{\mathrm{BIN}}(P)) = \mathrm{Hb}_{\mathrm{a}}(P)$.

To ease correctness considerations, we define for each non-binary rule $r \in P$, the translation $\mathrm{Tr}_{\mathrm{BIN}}(r)$ in (8), and an interpretation $I \subseteq \mathrm{Hb}(P)$, the set of *implied body atoms* $\mathrm{IBA}(r, I)$ which contains a^r_i from (8) whenever $0 < i < n$ and $\mathsf{b}_{i+1} \in I, \ldots, \mathsf{b}_n \in I$. For binary rules $r \in P$, $\mathrm{IBA}(r, I) = \emptyset$. Then we may define $\mathrm{IBA}(P, I) = \bigcup\{\mathrm{IBA}(r, I) \mid r \in P\}$ for a normal program P and an interpretation $I \subseteq \mathrm{Hb}(P)$. Note that the Herbrand base $\mathrm{Hb}(\mathrm{Tr}_{\mathrm{BIN}}(P))$ of the whole translation is obtained as $\mathrm{Hb}(P) \cup \mathrm{IBA}(P, \mathrm{Hb}(P))$.

LEMMA 27 ([JAN 03B]). — *Let P be a normal program and* $\mathrm{Tr}_{\mathrm{BIN}}(P)$ *its translation into a binary normal program.*

1) If $M_1 \subseteq M_2 \subseteq \mathrm{Hb}(P)$ and $M_1 \models P^{M_2}$, then $N_1 \models \mathrm{Tr}_{\mathrm{BIN}}(P)^{N_2}$ where $N_1 = M_1 \cup \mathrm{IBA}(P, M_1)$ and $N_2 = M_2 \cup \mathrm{IBA}(P, M_2)$.

2) If $N_1 \subseteq N_2 \subseteq \mathrm{Hb}(\mathrm{Tr}_{\mathrm{BIN}}(P))$ and $N_1 \models \mathrm{Tr}_{\mathrm{BIN}}(P)^{N_2}$, then $M_1 \subseteq M_2$ and $M_1 \models P^{M_2}$ hold for $M_1 = N_1 \cap \mathrm{Hb}(P)$ and $M_2 = N_2 \cap \mathrm{Hb}(P)$.

8. In practice, such atoms can be created using a numbering scheme that also takes the identity of r into account to avoid a clash with numbers introduced for other rules.

PROPOSITION 28 ([JAN 03B]). — *Let P be a normal logic program. If M is a stable model of P, then $N = M \cup \mathrm{IBA}(P, M)$ is a stable model of $\mathrm{Tr}_{\mathrm{BIN}}(P)$ such that $M = N \cap \mathrm{Hb}(P)$.*

PROPOSITION 29 ([JAN 03B]). — *Let P be a normal logic program. If N is a stable model of $\mathrm{Tr}_{\mathrm{BIN}}(P)$, then $M = N \cap \mathrm{Hb}(P)$ is a stable model of P such that $N = M \cup \mathrm{IBA}(P, M)$.*

THEOREM 30. — $\mathcal{P} \leq_{\mathrm{PFM}} \mathcal{B}$.

PROOF. — Let us begin with the faithfulness of $\mathrm{Tr}_{\mathrm{BIN}}$. It is clear by Definition 26 that $\mathrm{Hb}(P) \subseteq \mathrm{Hb}(\mathrm{Tr}_{\mathrm{BIN}}(P))$ and $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{BIN}}(P)) = \mathrm{Hb}_{\mathrm{v}}(P)$. By Proposition 28 there is an extension function $\mathrm{Ext}_{\mathrm{BIN}} : \mathrm{SM}(P) \rightarrow \mathrm{SM}(\mathrm{Tr}_{\mathrm{BIN}}(P))$ that maps $M \in \mathrm{SM}(P)$ into $N = \mathrm{Ext}_{\mathrm{BIN}}(M) = M \cup \mathrm{IBA}(P, M)$ included in $\mathrm{SM}(\mathrm{Tr}_{\mathrm{BIN}}(P))$ such that $M = N \cap \mathrm{Hb}(P)$. In addition to this, we know by Proposition 29 that if $N \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{BIN}}(P))$, then $M = N \cap \mathrm{Hb}(P) \in \mathrm{SM}(P)$ and $N = \mathrm{Ext}_{\mathrm{BIN}}(M)$. Thus $\mathrm{Tr}_{\mathrm{BIN}}$ is faithful by Proposition 20.

To establish modularity of $\mathrm{Tr}_{\mathrm{BIN}}$, let P and Q be two normal programs such that the module conditions M1–M4 from Definition 16 are satisfied. It is obvious by Definition 26 that $\mathrm{Tr}_{\mathrm{BIN}}(P \cup Q) = \mathrm{Tr}_{\mathrm{BIN}}(P) \cup \mathrm{Tr}_{\mathrm{BIN}}(Q)$. Let us then establish M1–M4.

(M1) Suppose that $\mathrm{Tr}_{\mathrm{BIN}}(P)$ and $\mathrm{Tr}_{\mathrm{BIN}}(Q)$ share a rule r. Two cases arise.

1) Suppose that $r \in P$. Then $|\mathrm{B}^+(r)| \leq 2$ holds by the definition of $\mathrm{Tr}_{\mathrm{BIN}}(P)$. Moreover, it follows that $r \notin Q$, as $P \cap Q = \emptyset$ by the module conditions. It follows that $r \notin \mathrm{Tr}_{\mathrm{BIN}}(Q)$, a contradiction.

2) Suppose that $r \in \mathrm{Tr}_{\mathrm{BIN}}(r')$ for some non-binary rule $r' \in P$. It follows by the module conditions that $r' \notin Q$. This means that no rule from $\mathrm{Tr}_{\mathrm{BIN}}(r')$ is included in $\mathrm{Tr}_{\mathrm{BIN}}(Q)$, since these rules are uniquely determined by new atoms $\mathsf{a}_1^{r'}, \ldots, \mathsf{a}_{n-1}^{r'}$ which depend on r', a contradiction.

It follows that $\mathrm{Tr}_{\mathrm{BIN}}(P) \cap \mathrm{Tr}_{\mathrm{BIN}}(Q) = \emptyset$.

(M2) Because P and Q satisfy M2, we know that $\mathrm{Hb}_{\mathrm{a}}(P) \cap \mathrm{Hb}_{\mathrm{a}}(Q) = \emptyset$. By Definition 26, these sets are preserved by $\mathrm{Tr}_{\mathrm{BIN}}$, i.e. $\mathrm{Hb}_{\mathrm{a}}(\mathrm{Tr}_{\mathrm{BIN}}(P)) = \mathrm{Hb}_{\mathrm{a}}(P)$ and $\mathrm{Hb}_{\mathrm{a}}(\mathrm{Tr}_{\mathrm{BIN}}(Q)) = \mathrm{Hb}_{\mathrm{a}}(Q)$. Thus $\mathrm{Hb}_{\mathrm{a}}(\mathrm{Tr}_{\mathrm{BIN}}(P)) \cap \mathrm{Hb}_{\mathrm{a}}(\mathrm{Tr}_{\mathrm{BIN}}(Q)) = \emptyset$.

(M3) Let us then assume that $\mathrm{Hb}_{\mathrm{h}}(\mathrm{Tr}_{\mathrm{BIN}}(P))$ and $\mathrm{Hb}(\mathrm{Tr}_{\mathrm{BIN}}(Q))$ share some atom a. Again, two cases arise.

1) Assume that $\mathsf{a} \in \mathrm{Hb}_{\mathrm{h}}(P)$. Since P and Q satisfy module conditions, we know that $\mathsf{a} \notin \mathrm{Hb}(Q)$. Since $\mathsf{a} \in \mathrm{Hb}(\mathrm{Tr}_{\mathrm{BIN}}(Q))$, the atom a must be one of the new atoms $\mathsf{a}_1^r, \ldots, \mathsf{a}_{n-1}^r$ associated with a non-binary rule $r \in Q$ with $|\mathrm{B}^+(r)| = n$. A contradiction by Definition 26, as $\mathsf{a} \in \mathrm{Hb}_{\mathrm{h}}(P)$ and P is also subject to translation as a module of $P \cup Q$.

2) Suppose that $\mathsf{a} \in \mathrm{Hb}_{\mathrm{h}}(\mathrm{Tr}_{\mathrm{BIN}}(P)) - \mathrm{Hb}_{\mathrm{h}}(P)$. Then a must be one of the new atoms $\mathsf{a}_1^r, \ldots, \mathsf{a}_{n-1}^r$ associated with a non-binary rule $r \in P$ with $|\mathrm{B}^+(r)| = n$. If $\mathsf{a} \in \mathrm{Hb}(Q)$, then a is not new, a contradiction. If $\mathsf{a} \in \mathrm{Hb}(\mathrm{Tr}_{\mathrm{BIN}}(Q)) - \mathrm{Hb}(Q)$, then

a must be one of the new atoms $\mathsf{a}_i^{r'}$ associated with a non-binary rule $r' \in Q$ with $|\mathrm{B}^+(r')| > 2$. Such atoms are different by Definition 26, a contradiction.

Thus $\mathrm{Hb}_\mathrm{h}(\mathrm{Tr}_\mathrm{BIN}(P)) \cap \mathrm{Hb}(\mathrm{Tr}_\mathrm{BIN}(Q)) = \emptyset$.

(M4) The last module condition follows by symmetry w.r.t the preceding one.

By Definition 26 and the modularity of Tr_BIN, the translation $\mathrm{Tr}_\mathrm{BIN}(P)$ of a normal program $P \in \mathcal{P}$ can be computed on a rule-by-rule basis. Moreover, the translation can be done in time linear in $||P||$, because (i) binary rules can be passed on unmodified and (ii) any non-binary rule (1) consisting of $2n + 3m + 2$ symbols is replaced by n rules (8) consisting of $(6 + 3m) + (n - 2) \times 6 + 4 = 6n + 3m - 2$ symbols, (iii) the atoms in $\mathrm{Hb}_\mathrm{a}(P)$ remain intact. ■

COROLLARY 31. — $\mathcal{B} =_\mathrm{PFM} \mathcal{P}$.

The main *intranslatability* result of this article follows: it is established that binary rules are not expressible in terms of unary rules even if we allow arbitrary number of negative literals in the bodies of rules or use an arbitrary number of unary rules.

THEOREM 32. — $\mathcal{B} \not\leq_\mathrm{FM} \mathcal{U}$.

PROOF. — Let us assume that there is a faithful and modular translation function Tr_UN from binary normal logic programs to unary ones. We intend to apply Tr_UN to a strictly binary normal logic program $B = \{\mathsf{a} \leftarrow \mathsf{b}, \mathsf{c};\ \mathsf{b} \leftarrow \mathsf{c}, \mathsf{a};\ \mathsf{c} \leftarrow \mathsf{a}, \mathsf{b}\}$ in conjunction with atomic programs $A_1 = \{\mathsf{a} \leftarrow\}$, $A_2 = \{\mathsf{b} \leftarrow\}$, and $A_3 = \{\mathsf{c} \leftarrow\}$. For these programs, $\mathrm{Hb}_\mathrm{v}(B) = \mathrm{Hb}(B) = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, $\mathrm{Hb}_\mathrm{v}(A_1) = \mathrm{Hb}(A_1) = \{\mathsf{a}\}$, $\mathrm{Hb}_\mathrm{v}(A_2) = \mathrm{Hb}(A_2) = \{\mathsf{b}\}$, and $\mathrm{Hb}_\mathrm{v}(A_3) = \mathrm{Hb}(A_3) = \{\mathsf{c}\}$. As there are no invisible atoms and the rules of the four programs are all distinct, the module conditions from Definition 16 are trivially satisfied.

Note that the rules of B essentially express that if any two of the atoms a, b, and c are inferable, then the third one should be, too. Thus each of the programs $B \cup A_1 \cup A_2$, $B \cup A_2 \cup A_3$, and $B \cup A_3 \cup A_1$ has a unique stable model $M = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. Since Tr_UN is faithful and modular, there are respective unique stable models

$$\begin{cases} N_1 = \mathrm{LM}(\mathrm{Tr}_\mathrm{UN}(B)^{N_1} \cup \mathrm{Tr}_\mathrm{UN}(A_1)^{N_1} \cup \mathrm{Tr}_\mathrm{UN}(A_2)^{N_1}) \\ N_2 = \mathrm{LM}(\mathrm{Tr}_\mathrm{UN}(B)^{N_2} \cup \mathrm{Tr}_\mathrm{UN}(A_2)^{N_2} \cup \mathrm{Tr}_\mathrm{UN}(A_3)^{N_2}) \\ N_3 = \mathrm{LM}(\mathrm{Tr}_\mathrm{UN}(B)^{N_3} \cup \mathrm{Tr}_\mathrm{UN}(A_3)^{N_3} \cup \mathrm{Tr}_\mathrm{UN}(A_1)^{N_3}) \end{cases} \tag{10}$$

of the translations $\mathrm{Tr}_\mathrm{UN}(B \cup A_1 \cup A_2)$, $\mathrm{Tr}_\mathrm{UN}(B \cup A_2 \cup A_3)$, and $\mathrm{Tr}_\mathrm{UN}(B \cup A_3 \cup A_1)$. These three stable models have to be assumed different, as the modules constituting the respective translations may involve invisible atoms and each of them is based on a different combination of modules.

Let us then turn our attention to the first equation in (10) and the "missing module" $\mathrm{Tr}_\mathrm{UN}(A_3)$. Note that A_3 has a unique stable model $M_3 = \{\mathsf{c}\}$. Let $N_3' = \mathrm{LM}(\mathrm{Tr}_\mathrm{UN}(A_3)^{N_3'})$ be the corresponding unique stable model of $\mathrm{Tr}_\mathrm{UN}(A_3)$, as implied by the faithfulness of Tr_UN. Note that $\mathrm{Tr}_\mathrm{UN}(B \cup A_1 \cup A_2)$ and $\mathrm{Tr}_\mathrm{UN}(A_3)$ satisfy the module conditions, as $B \cup A_1 \cup A_2$ and A_3 do and Tr_UN is modular. In

addition, $\mathrm{Hb_v}(\mathrm{Tr_{UN}}(B \cup A_1 \cup A_2)) \cap \mathrm{Hb_v}(\mathrm{Tr_{UN}}(A_3)) = \{\mathsf{c}\}$ and both N_1 and N_3' contain c so that we may apply Lemma 23 to conclude that $N_1 \cup N_3'$ is a stable model of $\mathrm{Tr_{UN}}(B \cup A_1 \cup A_2) \cup \mathrm{Tr_{UN}}(A_3)$ which equals to $\mathrm{Tr_{UN}}(B \cup A_1 \cup A_2 \cup A_3)$ by the modularity of $\mathrm{Tr_{UN}}$. Moreover, the reduct of the translation w.r.t. $N_1 \cup N_3'$ is $\mathrm{Tr_{UN}}(B \cup A_1 \cup A_2)^{N_1} \cup \mathrm{Tr_{UN}}(A_3)^{N_3'}$.

We let N_1' stand for the unique stable model of $\mathrm{Tr_{UN}}(A_1)$ which is guaranteed to exist by symmetry and which corresponds to the unique stable model $M_1 = \{\mathsf{a}\}$ of A_1. Similarly, let N_2' be the unique stable model corresponding to $M_2 = \{\mathsf{b}\}$. Then we may conclude that also $N_2 \cup N_1'$ and $N_3 \cup N_2'$ are stable models of $\mathrm{Tr_{UN}}(B \cup A_1 \cup A_2 \cup A_3)$ by using the last two equations of (10) concerning N_2 and N_3 in a symmetric fashion. On the other hand, M is the unique stable model of $B \cup A_1 \cup A_2 \cup A_3$, too. But then $\mathrm{Tr_{UN}}(B \cup A_1 \cup A_2 \cup A_3)$ must have a unique stable model — implying that $N_1 \cup N_3' = N_2 \cup N_1' = N_3 \cup N_2'$. Thus we may distinguish $N = N_1 \cap \mathrm{Hb}(\mathrm{Tr_{UN}}(B)) = N_2 \cap \mathrm{Hb}(\mathrm{Tr_{UN}}(B)) = N_3 \cap \mathrm{Hb}(\mathrm{Tr_{UN}}(B))$, and rewrite the preceding equalities as

$$\begin{cases} N \cup N_1' \cup N_2' = \mathrm{LM}(\mathrm{Tr_{UN}}(B)^N \cup \mathrm{Tr_{UN}}(A_1)^{N_1'} \cup \mathrm{Tr_{UN}}(A_2)^{N_2'}) \\ N \cup N_2' \cup N_3' = \mathrm{LM}(\mathrm{Tr_{UN}}(B)^N \cup \mathrm{Tr_{UN}}(A_2)^{N_2'} \cup \mathrm{Tr_{UN}}(A_3)^{N_3'}) \\ N \cup N_3' \cup N_1' = \mathrm{LM}(\mathrm{Tr_{UN}}(B)^N \cup \mathrm{Tr_{UN}}(A_3)^{N_3'} \cup \mathrm{Tr_{UN}}(A_1)^{N_1'}) \end{cases} \tag{11}$$

which still correspond to the unique stable models of $\mathrm{Tr_{UN}}(B \cup A_1 \cup A_2)$, $\mathrm{Tr_{UN}}(B \cup A_2 \cup A_3)$, and $\mathrm{Tr_{UN}}(B \cup A_3 \cup A_1)$, respectively. We proceed by reducing the first equation in (11) using Lemma 22. Note that $\mathrm{Tr_{UN}}(A_1) \cup \mathrm{Tr_{UN}}(A_2) = \mathrm{Tr_{UN}}(A_1 \cup A_2)$ by the modularity of $\mathrm{Tr_{UN}}$ and $\mathrm{Hb_v}(\mathrm{Tr_{UN}}(A_1 \cup A_2)) = \{\mathsf{a}, \mathsf{b}\} \subseteq \mathrm{Hb}(\mathrm{Tr_{UN}}(B))$. Thus we obtain $N = \mathrm{LM}(\mathrm{Tr_{UN}}(B)^N \cup \{\mathsf{a} \leftarrow;\ \mathsf{b} \leftarrow\})$ by Lemma 22. Recall that N contains c in addition to a and b. Let $\mathrm{Tr_{UN}}(B)_0^N$ and $\mathrm{Tr_{UN}}(B)_1^N$ denote the disjoint sets of atomic and strictly unary rules of $\mathrm{Tr_{UN}}(B)^N$, respectively. It follows by Lemma 24 that there is an atomic rule $\mathsf{d} \leftarrow$ in $\mathrm{Tr_{UN}}(B)_0^N \cup \{\mathsf{a} \leftarrow;\ \mathsf{b} \leftarrow\}$ such that $\mathsf{c} \in \mathrm{LM}(\mathrm{Tr_{UN}}(B)_1^N \cup \{\mathsf{d} \leftarrow\})$. As $\mathrm{LM}(\cdot)$ is a monotonic operator, we obtain two cases: $\mathsf{c} \in \mathrm{LM}(\mathrm{Tr_{UN}}(B)^N \cup \{\mathsf{a} \leftarrow\})$ *or* $\mathsf{c} \in \mathrm{LM}(\mathrm{Tr_{UN}}(B)^N \cup \{\mathsf{b} \leftarrow\})$.

In the first case, we obtain $\mathsf{c} \in \mathrm{LM}(\mathrm{Tr_{UN}}(B)^N \cup \mathrm{Tr_{UN}}(A_1)^{N_1'})$ by the monotonicity of the operator $\mathrm{LM}(\cdot)$ again. Recall that $\mathrm{Hb_v}(\mathrm{Tr_{UN}}(A_3)) = \mathrm{Hb_v}(A_3) = \{\mathsf{c}\}$ by the definition of $\mathrm{Tr_{UN}}$. As a result of applying Lemma 22 to the third equation in (11), $\mathrm{Tr_{UN}}(A_3)^{N_3'}$ is reduced to $\{\mathsf{c} \leftarrow\}$, so that $N \cup N_1' = \mathrm{LM}(\mathrm{Tr_{UN}}(B)^N \cup \mathrm{Tr_{UN}}(A_1)^{N_1'} \cup \{\mathsf{c} \leftarrow\})$. Since c belongs to $\mathrm{LM}(\mathrm{Tr_{UN}}(B)^N \cup \mathrm{Tr_{UN}}(A_1)^{N_1'})$ this simplifies to $N \cup N_1' = \mathrm{LM}(\mathrm{Tr_{UN}}(B)^N \cup \mathrm{Tr_{UN}}(A_1)^{N_1'})$. Because $N \subseteq \mathrm{Hb}(\mathrm{Tr_{UN}}(B))$, $N_1' \subseteq \mathrm{Hb}(\mathrm{Tr_{UN}}(A_1))$, and N and N_1' coincide on the atoms in $\mathrm{Hb_v}(\mathrm{Tr_{UN}}(B)) \cap \mathrm{Hb_v}(\mathrm{Tr_{UN}}(A_1)) = \{\mathsf{a}\}$, we get $\mathrm{Tr_{UN}}(B)^N = \mathrm{Tr_{UN}}(B)^{N \cup N_1'}$ and $\mathrm{Tr_{UN}}(A_1)^{N_1'} = \mathrm{Tr_{UN}}(A_1)^{N \cup N_1'}$. Then the modularity of $\mathrm{Tr_{UN}}$ implies $N \cup N_1' = \mathrm{LM}(\mathrm{Tr_{UN}}(B \cup A_1)^{N \cup N_1'})$. Thus $\mathrm{Tr_{UN}}(B \cup A_1)$ possesses a stable model $N \cup N_1'$ containing $\{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. A contradiction, since $B \cup A_1$ has a unique stable model $\{\mathsf{a}\}$.

In the second case, we can analogously construct a stable model $N \cup N_2'$ for $\mathrm{Tr_{UN}}(B \cup A_2)$ using the second equation in (11). Again, this is a contradiction, as $\{\mathsf{a}, \mathsf{b}, \mathsf{c}\} \subseteq N \cup N_2'$ and $\{\mathsf{b}\}$ is the unique stable model of $B \cup A_2$. ■

COROLLARY 33. — *$\mathcal{B} \not\leq_{\mathrm{PFM}} \mathcal{U}$, $\mathcal{U} <_{\mathrm{PFM}} \mathcal{B}$, and $\mathcal{U} <_{\mathrm{PFM}} \mathcal{P}$.*

It remains to explore the strictness of the relationship $\mathcal{A} \leq_{\mathrm{PFM}} \mathcal{U}$. At this point, it is worth demonstrating a particular translation technique [SCH 95, Proof of Theorem 3.10], which suitably exploits new atoms and negative literals and thus serves as a potential candidate for a faithful and modular translation function from $\mathcal{U}$ to $\mathcal{A}$.

EXAMPLE 34. — Consider programs $P_1 = \{\mathsf{a} \leftarrow \mathsf{b}\}$ and $P_2 = \{\mathsf{b} \leftarrow \mathsf{c}\}$ and a translation of $P_1 \cup P_2$ into an atomic normal logic program $\mathrm{Tr}_{\mathrm{SCH}}(P_1 \cup P_2) = \mathrm{Tr}_{\mathrm{SCH}}(P_1) \cup \mathrm{Tr}_{\mathrm{SCH}}(P_2) = \{\mathsf{a} \leftarrow \sim\overline{\mathsf{b}};\ \overline{\mathsf{b}} \leftarrow \sim\mathsf{b}\} \cup \{\mathsf{b} \leftarrow \sim\overline{\mathsf{c}};\ \overline{\mathsf{c}} \leftarrow \sim\mathsf{c}\}$ where the intuitive reading of the new atoms $\overline{\mathsf{b}}$ and $\overline{\mathsf{c}}$ is that b and c are false, respectively. The translation tries to capture the rules of P using a kind of double negation. In particular, the rules $\overline{\mathsf{b}} \leftarrow \sim\mathsf{b}$ and $\overline{\mathsf{c}} \leftarrow \sim\mathsf{c}$ can be understood to encode the standard closed world assumption [REI 78]: according to these rules b and c are false by default.

Let us then analyze the behavior of $P_1 \cup P_2$ and $\mathrm{Tr}_{\mathrm{SCH}}(P_1 \cup P_2)$ when they are placed in the context of $A_0 = \emptyset$, $A_1 = \{\mathsf{a} \leftarrow\}$, $A_2 = \{\mathsf{b} \leftarrow\}$, and $A_3 = \{\mathsf{c} \leftarrow\}$. The programs $P_1 \cup P_2 \cup A_i$ where $i \in \{0, 1, 2, 3\}$ have unique stable models $M_0 = \emptyset$, $M_1 = \{\mathsf{a}\}$, $M_2 = \{\mathsf{a}, \mathsf{b}\}$, and $M_3 = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$, respectively. Accordingly, the translations $\mathrm{Tr}_{\mathrm{SCH}}(P_1 \cup P_2 \cup A_i)$ where $i \in \{0, 1, 2, 4\}$ have unique stable models $N_0 = \{\overline{\mathsf{b}}, \overline{\mathsf{c}}\}$, $N_1 = \{\mathsf{a}, \overline{\mathsf{b}}, \overline{\mathsf{c}}\}$, $N_2 = \{\mathsf{a}, \mathsf{b}, \overline{\mathsf{c}}\}$, and $N_3 = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$. □

The translation $\mathrm{Tr}_{\mathrm{SCH}}(P_1 \cup P_2)$ seems to capture the essentials of $P_1 \cup P_2$ in a modular and faithful manner. However, severe problems arise with programs containing an inferential loop that lets one to infer a from a, for instance. The simplest possible example of this kind is $P = \{\mathsf{a} \leftarrow \mathsf{a}\}$ having a minimal model $\mathrm{LM}(P) = \emptyset$. But the translation $\mathrm{Tr}_{\mathrm{SCH}}(P) = \{\mathsf{a} \leftarrow \sim\overline{\mathsf{a}};\ \overline{\mathsf{a}} \leftarrow \sim\mathsf{a}\}$ has two stable models $\{\overline{\mathsf{a}}\}$ and $\{\mathsf{a}\}$. The former stable model is what we would expect on the basis of Example 34, but the latter is spurious — dashing our hopes for $\mathrm{Tr}_{\mathrm{SCH}}$ being faithful and modular in general. Next we prove that the problems with $\mathrm{Tr}_{\mathrm{SCH}}$ cannot be settled.

THEOREM 35. — $\mathcal{U} \not\leq_{\mathrm{FM}} \mathcal{A}$.

PROOF. — Suppose there is a faithful and modular translation function $\mathrm{Tr}_{\mathrm{AT}}$ from $\mathcal{U}$ to $\mathcal{A}$. Then we analyze two unary normal programs $U_1 = \{\mathsf{a} \leftarrow \mathsf{b}\}$ and $U_2 = \{\mathsf{b} \leftarrow \mathsf{a}\}$, their combinations with atomic normal programs $A_1 = \{\mathsf{b} \leftarrow\}$ and $A_2 = \{\mathsf{a} \leftarrow\}$, and their translations under $\mathrm{Tr}_{\mathrm{AT}}$. To check the module conditions, we note that $\mathrm{Hb}_{\mathrm{v}}(U_1) = \mathrm{Hb}(U_1) = \{\mathsf{a}, \mathsf{b}\} = \mathrm{Hb}(U_2) = \mathrm{Hb}_{\mathrm{v}}(U_1)$, $\mathrm{Hb}_{\mathrm{v}}(A_1) = \mathrm{Hb}(A_1) = \{\mathsf{b}\}$, and $\mathrm{Hb}_{\mathrm{v}}(A_2) = \mathrm{Hb}(A_2) = \{\mathsf{a}\}$. Because there are no hidden atoms and the rules of the four programs are distinct, the module conditions are trivially satisfied by U_1 and A_1, by U_2 and A_2, by $U_1 \cup A_1$ and $U_2 \cup A_2$, and by $U_1 \cup U_2$ and $A_1 \cup A_2$.

The program $U_1 \cup A_1$ has a unique stable model $M_1 = \mathrm{LM}(U_1 \cup A_1) = \{\mathsf{a}, \mathsf{b}\}$. The translation $\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup A_1) = \mathrm{Tr}_{\mathrm{AT}}(U_1) \cup \mathrm{Tr}_{\mathrm{AT}}(A_1)$ and the modularity of $\mathrm{Tr}_{\mathrm{AT}}$ implies $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup A_1)) = \mathrm{Hb}_{\mathrm{v}}(U_1 \cup A_1) = \{\mathsf{a}, \mathsf{b}\}$. Since $\mathrm{Tr}_{\mathrm{AT}}$ is also faithful, the translation $\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup A_1)$ has a unique stable model $N_1 = \mathrm{LM}(\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup A_1)^{N_1}) = \mathrm{LM}(\mathrm{Tr}_{\mathrm{AT}}(U_1)^{N_1} \cup \mathrm{Tr}_{\mathrm{AT}}(A_1)^{N_1})$ such that $\{\mathsf{a}, \mathsf{b}\} \subseteq N_1$. Then it holds by symmetry that $M_2 = \mathrm{LM}(U_2 \cup A_2) = \{\mathsf{a}, \mathsf{b}\}$ is the unique stable model of $U_2 \cup A_2$ and $N_2 = \mathrm{LM}(\mathrm{Tr}_{\mathrm{AT}}(U_2 \cup A_2)^{N_2})$ is the unique stable model of the translation $\mathrm{Tr}_{\mathrm{AT}}(U_2 \cup$

$A_2) = \mathrm{Tr}_{\mathrm{AT}}(U_2) \cup \mathrm{Tr}_{\mathrm{AT}}(A_2)$, for which $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(U_2 \cup A_2)) = \mathrm{Hb}_{\mathrm{v}}(U_2 \cup A_2) = \{\mathsf{a}, \mathsf{b}\}$ holds, so that $\{\mathsf{a}, \mathsf{b}\} \subseteq N_2$.

Recall that $\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup A_1) = \mathrm{Tr}_{\mathrm{AT}}(U_1) \cup \mathrm{Tr}_{\mathrm{AT}}(A_1)$ is an atomic program and $\mathsf{a} \in \mathrm{LM}(\mathrm{Tr}_{\mathrm{AT}}(U_1)^{N_1} \cup \mathrm{Tr}_{\mathrm{AT}}(A_1)^{N_1})$. Thus $\mathsf{a} \leftarrow$ belongs to the reduct by Lemma 25. Since $\mathsf{a} \notin \mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(A_1))$ and $\mathsf{a} \in \mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(U_1))$ by the faithfulness of $\mathrm{Tr}_{\mathrm{AT}}$, and the translations $\mathrm{Tr}_{\mathrm{AT}}(U_1)$ and $\mathrm{Tr}_{\mathrm{AT}}(A_1)$ satisfy module conditions by the modularity of $\mathrm{Tr}_{\mathrm{AT}}$, we have $\mathsf{a} \notin \mathrm{Hb}(\mathrm{Tr}_{\mathrm{AT}}(A_1))$. Thus $\mathsf{a} \leftarrow$ cannot belong to $\mathrm{Tr}_{\mathrm{AT}}(A_1)^{N_1}$. So it must belong to $\mathrm{Tr}_{\mathrm{AT}}(U_1)^{N_1}$ and $\mathsf{b} \leftarrow$ belongs to $\mathrm{Tr}_{\mathrm{AT}}(U_2)^{N_2}$ by symmetry.

Because $U_1 \cup A_1$ and $U_2 \cup A_2$ satisfy the module conditions, and N_1 and N_2 coincide up to the atoms in $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup A_2)) \cap \mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(U_2 \cup A_2)) = \{\mathsf{a}, \mathsf{b}\}$, we know by Lemma 23 that $N_1 \cup N_2$ is a stable model of $\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup A_1) \cup \mathrm{Tr}_{\mathrm{AT}}(U_2 \cup A_2)$ which equals to $\mathrm{Tr}_{\mathrm{AT}}(U_1) \cup \mathrm{Tr}_{\mathrm{AT}}(A_1) \cup \mathrm{Tr}_{\mathrm{AT}}(U_2) \cup \mathrm{Tr}_{\mathrm{AT}}(A_2) = \mathrm{Tr}_{\mathrm{AT}}(U_1 \cup U_2) \cup \mathrm{Tr}_{\mathrm{AT}}(A_1 \cup A_2)$ by the modularity of $\mathrm{Tr}_{\mathrm{AT}}$. Moreover, the reduct $(\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup A_1) \cup \mathrm{Tr}_{\mathrm{AT}}(U_2 \cup A_2))^{N_1 \cup N_2}$ is the union of $\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup A_1)^{N_1}$ and $\mathrm{Tr}_{\mathrm{AT}}(U_2 \cup A_2)^{N_2}$, i.e. $\mathrm{Tr}_{\mathrm{AT}}(U_1)^{N_1} \cup \mathrm{Tr}_{\mathrm{AT}}(U_2)^{N_2} \cup \mathrm{Tr}_{\mathrm{AT}}(A_1)^{N_1} \cup \mathrm{Tr}_{\mathrm{AT}}(A_2)^{N_2}$.

Since the visible parts $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(A_1)) = \{\mathsf{b}\}$ and $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(A_2)) = \{\mathsf{a}\}$ are contained in $\mathrm{Hb}(\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup U_2))$, it follows by Lemma 22 that the projection $N = (N_1 \cup N_2) \cap \mathrm{Hb}(\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup U_2)) = \mathrm{LM}(\mathrm{Tr}_{\mathrm{AT}}(U_1)^{N_1} \cup \mathrm{Tr}_{\mathrm{AT}}(U_2)^{N_2} \cup \{\mathsf{a} \leftarrow;\ \mathsf{b} \leftarrow\})$. Since $\mathsf{a} \leftarrow$ belongs to $\mathrm{Tr}_{\mathrm{AT}}(U_1)^{N_1}$ and $\mathsf{b} \leftarrow$ to $\mathrm{Tr}_{\mathrm{AT}}(U_2)^{N_2}$, we can establish that $N = \mathrm{LM}(\mathrm{Tr}_{\mathrm{AT}}(U_1)^{N_1} \cup \mathrm{Tr}_{\mathrm{AT}}(U_2)^{N_2})$. Moreover, the equality $\mathrm{Tr}_{\mathrm{AT}}(U_1)^{N_1} \cup \mathrm{Tr}_{\mathrm{AT}}(U_2)^{N_2} = \mathrm{Tr}_{\mathrm{AT}}(U_1 \cup U_2)^{N}$ follows by the definition of N and the fact that N_1 and N_2 coincide on the atoms contained in $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(U_1)) = \mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(U_2)) = \{\mathsf{a}, \mathsf{b}\}$. Thus $N = \mathrm{LM}(\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup U_2)^{N})$ is a stable model of $\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup U_2)$.

Since $\mathrm{Hb}_{\mathrm{v}}(U_1 \cup U_2) = \mathrm{Hb}(U_1 \cup U_2) = \{\mathsf{a}, \mathsf{b}\}$ and $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup U_2)) = \mathrm{Hb}_{\mathrm{v}}(U_1 \cup U_2)$ by the faithfulness of $\mathrm{Tr}_{\mathrm{AT}}$, and $N \cap \mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(U_1 \cup U_2)) = \{\mathsf{a}, \mathsf{b}\}$, the faithfulness of $\mathrm{Tr}_{\mathrm{AT}}$ implies that $U_1 \cup U_2$ has a stable model $M = \{\mathsf{a}, \mathsf{b}\}$. A contradiction, as $\emptyset$ is the unique stable model of $U_1 \cup U_2$. ■

COROLLARY 36. — *$\mathcal{U} \not\leq_{\mathrm{PFM}} \mathcal{A}$, $\mathcal{A} <_{\mathrm{PFM}} \mathcal{U}$, $\mathcal{A} <_{\mathrm{PFM}} \mathcal{B}$, and $\mathcal{A} <_{\mathrm{PFM}} \mathcal{P}$.*

Corollary 36 completes our view as regards the mutual relationships of $\mathcal{A}$, $\mathcal{U}$, $\mathcal{B}$, and $\mathcal{P}$. However, we may continue the analysis by comparing the respective classes of positive programs with them. To this end, we recall $\mathcal{C}^+ \leq_{\mathrm{PFM}} \mathcal{C}$ holds by Proposition 19 for any of the classes $\mathcal{C}$ under consideration. Strictness follows quite easily.

THEOREM 37. — *For any $\mathcal{C} \in \{\mathcal{A}, \mathcal{U}, \mathcal{B}, \mathcal{P}\}$, $\mathcal{C} \not\leq_{\mathrm{F}} \mathcal{C}^+$.*

PROOF. — Let us assume that there is a faithful translation function Tr from $\mathcal{C}$ to $\mathcal{C}^+$. Consider a logic program $P = \{\mathsf{a} \leftarrow \sim\mathsf{a}\}$ which serves as a representative of the class $\mathcal{C}$. Let Q be the translation $\mathrm{Tr}(P)$ in $\mathcal{C}^+$. Now P has no stable models, but the translation Q has a unique stable model $\mathrm{LM}(Q)$. A contradiction, as Tr is faithful. ■

COROLLARY 38. — *For any $\mathcal{C} \in \{\mathcal{A}, \mathcal{U}, \mathcal{B}, \mathcal{P}\}$, $\mathcal{C} \not\leq_{\mathrm{PFM}} \mathcal{C}^+$ and $\mathcal{C}^+ <_{\mathrm{PFM}} \mathcal{C}$.*

Finally, let us conclude the mutual relationships of the positive classes. As before, we obtain $\mathcal{A}^+ \leq_{\mathrm{PFM}} \mathcal{U}^+ \leq_{\mathrm{PFM}} \mathcal{B}^+ \leq_{\mathrm{PFM}} \mathcal{P}^+$ immediately by Proposition 19. On

the other hand, Theorems 30, 32 and 35 specialize to the case of positive programs although slightly simpler counter-examples could be used as done in [JAN 03b].

THEOREM 39. — $\mathcal{P}^+ \leq_{\mathrm{PFM}} \mathcal{B}^+$, $\mathcal{B}^+ \not\leq_{\mathrm{FM}} \mathcal{U}^+$, *and* $\mathcal{U}^+ \not\leq_{\mathrm{FM}} \mathcal{A}^+$.

COROLLARY 40. — $\mathcal{A}^+ <_{\mathrm{PFM}} \mathcal{U}^+$, $\mathcal{U}^+ <_{\mathrm{PFM}} \mathcal{B}^+$, *and* $\mathcal{B}^+ =_{\mathrm{PFM}} \mathcal{P}^+$.

The relationships established so far give rise to the *expressive power hierarchy* (EPH) of logic programs which is illustrated in Figure 1. To conclude, the classes of the hierarchy indicate that the number of positive body literals can be limited to two without an effective loss of expressive power (recall $\mathrm{Tr}_{\mathrm{BIN}}$ from Section 4.2). It is easy to inspect that the proof of Theorem 37 remains valid even if we consider weaker notions of faithfulness based on the equivalence relations $\equiv_{\mathrm{vs}}$, $\equiv_{\mathrm{vb}}$, and $\equiv_{\mathrm{vc}}$ addressed in Proposition 13. Thus the gap between a positive class $\mathcal{C}^+$ and the respective class $\mathcal{C}$ remains intact for all $\mathcal{C} \in \{\mathcal{A}, \mathcal{U}, \mathcal{B}, \mathcal{P}\}$. Quite similarly, the relationships in the lower end of the hierarchy are not affected under $\equiv_{\mathrm{vs}}$ and $\equiv_{\mathrm{vb}}$ since positive programs have unique stable models and thus $\equiv_{\mathrm{v}}$, $\equiv_{\mathrm{vs}}$, and $\equiv_{\mathrm{vb}}$ coincide. For the same reason $\equiv_{\mathrm{vc}}$ is uninteresting and the respective notion of faithfulness would equate all positive classes. Finally, we emphasize that the strict relationships $\mathcal{A} <_{\mathrm{PFM}} \mathcal{U}$ and $\mathcal{U} <_{\mathrm{PFM}} \mathcal{B}$ may cease to hold under weaker notions of faithfulness but we leave the analysis as future work. The counter-examples of Theorems 32 and 35 cover the notion of Definition 15 which we consider as the most appropriate one for ASP.

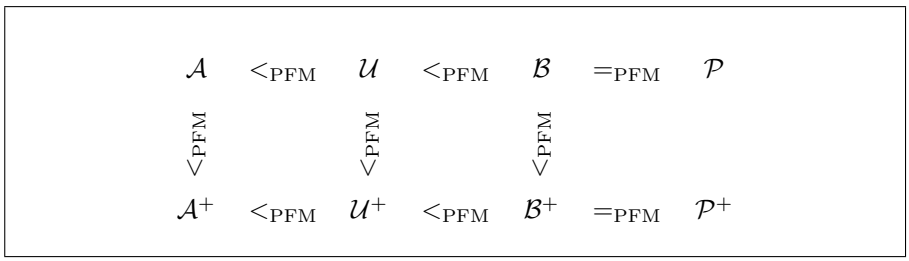

Figure 1. *Expressive power hierarchy (EPH) based on polynomial, faithful and modular (PFM) translation functions*

4.3. *Comparison with propositional logic*

Let us now concentrate on the case of propositional logic analyze its expressive power within our framework. It is assumed that a propositional theory S is given as a set of clauses of the form (3).[9] The fundamental *satisfiability problem* (SAT) is about checking if a given set of clauses S has a model in the classical sense. However, in this article, we are interested in all models of S rather than checking the existence of

9. Any propositional theory can be transformed into clausal form and the transformation is linear if new atoms can be introduced [TSE 83]. Without new atoms it takes a lot of space to transform formulas like $(\mathsf{a}_1 \wedge \mathsf{b}_1) \vee \cdots \vee (\mathsf{a}_n \wedge \mathsf{b}_n)$ into clausal form.

a model. The reason is that we assume in analogy to ASP that the models of a set of clauses S correspond to the solutions of the problem formalized as S. It is possible to capture the models of a set of clauses S with the stable models of a translation of S into a normal program. In fact, we can do this using only atomic rules. The basic idea is as follows. The rules $\mathsf{a} \leftarrow \sim\overline{\mathsf{a}}$ and $\overline{\mathsf{a}} \leftarrow \sim\mathsf{a}$ are needed to select the truth value of each atom $\mathsf{a} \in \mathrm{Hb}(S)$. Here $\overline{\mathsf{a}} \notin \mathrm{Hb}(S)$ is a new atom meaning that a is false (c.f. Example 34). Given these rules, we obtain all model candidates for S as the stable models of the rules. Yet we have to ensure that every clause (3) is satisfied. This is accomplished by introducing a new atom $\mathsf{f} \notin \mathrm{Hb}(S)$ and an atomic rule

$$\mathsf{f} \leftarrow \sim\mathsf{f}, \sim\mathsf{a}_1, \ldots, \sim\mathsf{a}_n, \sim\overline{\mathsf{b}}_1, \ldots, \sim\overline{\mathsf{b}}_m \tag{12}$$

for each clause (3) in S. These kinds of rules exclude model candidates in which some of the clauses is false. If the full syntax of normal programs is assumed, then it would be more intuitive to use a rule of the form $\mathsf{f} \leftarrow \mathsf{b}_1, \ldots, \mathsf{b}_m, \sim\mathsf{f}, \sim\mathsf{a}_1, \ldots, \sim\mathsf{a}_n$ which is not atomic, but "double negation" is needed in order to make the rule atomic. Yet another technique is given in [NIE 99]: a new atom c is introduced for each clause (3) which is translated into $\mathsf{c} \leftarrow \mathsf{a}_1;\ \ldots;\ \mathsf{c} \leftarrow \mathsf{a}_n;\ \mathsf{c} \leftarrow \overline{\mathsf{b}_1};\ \ldots;\ \mathsf{c} \leftarrow \overline{\mathsf{b}_m}$. Then (12) can be replaced by $\mathsf{f} \leftarrow \sim\mathsf{f}, \sim\mathsf{c}$. However, to meet the module conditions from Definition 16, we have to localize the choice of truth values. For this reason we translate a clause c of the form (3) into a set of rules

$$\begin{aligned}\mathrm{Tr}_{\mathrm{LP}}(c) = {} & \{\mathsf{f}^c \leftarrow \sim\mathsf{f}^c, \sim\mathsf{a}_1, \ldots, \sim\mathsf{a}_n, \sim\mathsf{b}_1^c, \ldots, \sim\mathsf{b}_m^c\} \cup \\ & \{\mathsf{a}_i \leftarrow \sim\mathsf{a}_i^c;\ \ \mathsf{a}_i^c \leftarrow \sim\mathsf{a}_i \mid 0 < i \leq n\} \cup \\ & \{\mathsf{b}_i \leftarrow \sim\mathsf{b}_i^c;\ \ \mathsf{b}_i^c \leftarrow \sim\mathsf{b}_i \mid 0 < i \leq m\}\end{aligned} \tag{13}$$

where f^c, $\mathsf{a}_1^c, \ldots, \mathsf{a}_n^c$, and $\mathsf{b}_1^c, \ldots, \mathsf{b}_m^c$ are new atoms that are unique to c. This implies that the choice of the truth value of an atom a is shared by the rules in which the atom appears. However, these choices are synchronized, as a is shared among the rules, and this is how a is assigned a unique truth value.

DEFINITION 41. — *A set of clauses S is translated into*

$$\mathrm{Tr}_{\mathrm{LP}}(S) = \bigcup\{\mathrm{Tr}_{\mathrm{LP}}(c) \mid c \in S\} \cup \{\mathsf{a} \leftarrow \sim\overline{\mathsf{a}};\ \ \overline{\mathsf{a}} \leftarrow \sim\mathsf{a} \mid \mathsf{a} \in \mathrm{Hb}_{\mathrm{a}}(S)\}$$

with $\mathrm{Hb}_{\mathrm{a}}(\mathrm{Tr}_{\mathrm{LP}}(S)) = \emptyset$, $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{LP}}(S)) = \mathrm{Hb}_{\mathrm{v}}(S)$, *and* $\mathrm{Hb}_{\mathrm{h}}(\mathrm{Tr}_{\mathrm{LP}}(S)) = \mathrm{Hb}_{\mathrm{h}}(S) \cup \{\mathsf{f}^c \mid c \in S\} \cup \{\mathsf{a}^c \mid c \in S$ *and* a *appears in* $c\} \cup \{\overline{\mathsf{a}} \mid \mathsf{a} \in \mathrm{Hb}_{\mathrm{a}}(S)\}$.

A particular feature of the translation $\mathrm{Tr}_{\mathrm{LP}}(S)$ is that all atoms of $\mathrm{Hb}(S)$ actually appear in the rules of $\mathrm{Tr}_{\mathrm{LP}}(S)$ and thus $\mathrm{Hb}_{\mathrm{a}}(\mathrm{Tr}_{\mathrm{LP}}(S))$ remains empty. The rules associated with the atoms in $\mathrm{Hb}_{\mathrm{a}}(S)$ are necessary in order to capture the classical models of S properly, since $\mathrm{Hb}(S)$ may contain atoms that do not appear in the clauses of S; and according to Section 2.3 classical models of S are subsets of $\mathrm{Hb}(S)$. Given a set of clauses S, an interpretation $M \subseteq \mathrm{Hb}(S)$, and a clause $c \in S$, we define the set of *complementary atoms* $\mathrm{CA}(c, M)$ which contains a^c whenever a appears in c and $\mathsf{a} \notin M$. For the set S as whole, we let

$$\mathrm{CA}(S, M) = \bigcup\{\mathrm{CA}(c, M) \mid c \in S\} \cup \{\overline{\mathsf{a}} \mid \mathsf{a} \in \mathrm{Hb}_{\mathrm{a}}(S) - M\} \tag{14}$$

which takes also the additional atoms from $\mathrm{Hb}_{\mathrm{a}}(S)$ properly into account. We are now ready to address the correctness of the translation function $\mathrm{Tr}_{\mathrm{LP}}$.

PROPOSITION 42 ([JAN 03B]). — *Let S be a set of clauses. If $M \subseteq \mathrm{Hb}(S)$ is a classical model of S, then $N = M \cup \mathrm{CA}(S, M)$ is a stable model of $\mathrm{Tr}_{\mathrm{LP}}(S)$ such that $N \cap \mathrm{Hb}(S) = M$.*

PROPOSITION 43 ([JAN 03B]). — *Let S be a set of clauses. If $N \subseteq \mathrm{Hb}(\mathrm{Tr}_{\mathrm{LP}}(S))$ is a stable model of $\mathrm{Tr}_{\mathrm{LP}}(S)$, then $M = N \cap \mathrm{Hb}(S) \models S$ and $N = M \cup \mathrm{CA}(S, M)$.*

THEOREM 44 ([JAN 03B]). — *$\mathcal{PT} \leq_{\mathrm{PFM}} \mathcal{A}$.*

On the other hand, it is impossible to translate an atomic normal program P into a set of clauses in a faithful and modular way. This result has been established by Niemelä [NIE 99, Proposition 4.3] for normal programs in general, but different notions of faithfulness and modularity are employed in Niemelä's proof.

THEOREM 45. — *$\mathcal{A} \not\leq_{\mathrm{FM}} \mathcal{PT}$.*

PROOF. — Let us assume that there exists a faithful and modular translation function $\mathrm{Tr} : \mathcal{A} \to \mathcal{PT}$. Then consider atomic normal logic programs $P_1 = \{\mathsf{a} \leftarrow \sim\mathsf{a}\}$ and $P_2 = \{\mathsf{a} \leftarrow\}$ with $\mathrm{Hb}_{\mathrm{v}}(P_1) = \mathrm{Hb}(P_1) = \mathrm{Hb}_{\mathrm{v}}(P_2) = \mathrm{Hb}(P_2) = \{\mathsf{a}\}$. It is clear that P_1 and P_2 satisfy the module conditions from Definition 16. The program P_1 has no stable models while P_2 has a unique stable model $M = \{\mathsf{a}\}$. Since Tr is faithful, the translation $\mathrm{Tr}(P_1)$ must be propositionally inconsistent. By the modularity of Tr, the translation $\mathrm{Tr}(P_1 \cup P_2) = \mathrm{Tr}(P_1) \cup \mathrm{Tr}(P_2)$ which is also propositionally inconsistent, i.e. has no models. But this contradicts the faithfulness of Tr, since M is also the unique stable model of $P_1 \cup P_2$. Hence there is no such Tr. ■

COROLLARY 46. — *$\mathcal{PT} <_{\mathrm{PFM}} \mathcal{C}$ holds for any $\mathcal{C} \in \{\mathcal{A}, \mathcal{U}, \mathcal{B}, \mathcal{P}\}$.*

THEOREM 47. — *$\mathcal{PT} \not\leq_{\mathrm{F}} \mathcal{C}^+$ holds for any $\mathcal{C} \in \{\mathcal{A}, \mathcal{U}, \mathcal{B}, \mathcal{P}\}$.*

PROOF. — The set of clauses $S = \{\{\mathsf{a}, \neg\mathsf{a}\}\}$ has two classical models $M_1 = \emptyset$ and $M_2 = \{\mathsf{a}\}$ which cannot be faithfully captured by a positive program $P = \mathrm{Tr}(S)$ possessing a unique stable model $\mathrm{LM}(P)$. ■

THEOREM 48. — *$\mathcal{C}^+ \not\leq_{\mathrm{FM}} \mathcal{PT}$ holds for all classes $\mathcal{C}^+ \in \{\mathcal{A}^+, \mathcal{U}^+, \mathcal{B}^+, \mathcal{P}^+\}$.*

PROOF. — Suppose there is a faithful and modular translation function $\mathrm{Tr}_{\mathrm{CL}}$ from $\mathcal{C}^+$ to $\mathcal{PT}$. Consider programs $P_1 = \{\mathsf{a} \leftarrow\}$ and $P_2 = \emptyset$ with $\mathrm{Hb}_{\mathrm{v}}(P_i) = \mathrm{Hb}(P_i) = \{\mathsf{a}\}$ for $i \in \{1, 2\}$. Then $\mathrm{Hb}_{\mathrm{a}}(P_2) = \{\mathsf{a}\}$ and module conditions are satisfied. Now P_1 and P_2 have unique stable models $\mathrm{LM}(P_1) = \{\mathsf{a}\}$ and $\mathrm{LM}(P_2) = \emptyset$, respectively. It follows by the faithfulness of $\mathrm{Tr}_{\mathrm{CL}}$ that the unique classical models of $\mathrm{Tr}_{\mathrm{CL}}(P_1)$ and $\mathrm{Tr}_{\mathrm{CL}}(P_2)$ are N_1 and N_2 such that $M_1 = N_1 \cap \{\mathsf{a}\}$ and $M_2 = N_2 \cap \{\mathsf{a}\}$. It follows that $\mathrm{Tr}_{\mathrm{CL}}(P_1) \models \mathsf{a}$ and $\mathrm{Tr}_{\mathrm{CL}}(P_2) \models \neg\mathsf{a}$. On the other hand, we know by the modularity of $\mathrm{Tr}_{\mathrm{CL}}$ that $\mathrm{Tr}_{\mathrm{CL}}(P_1 \cup P_2) = \mathrm{Tr}_{\mathrm{CL}}(P_1) \cup \mathrm{Tr}_{\mathrm{CL}}(P_2)$. It follows that $\mathrm{Tr}_{\mathrm{CL}}(P_1 \cup P_2) \models \mathsf{a} \wedge \neg\mathsf{a}$, i.e. $\mathrm{Tr}_{\mathrm{CL}}(P_1 \cup P_2)$ has no models. However, the program $P_1 \cup P_2$ has a unique stable model $\mathrm{LM}(P_1 \cup P_2) = \{\mathsf{a}\}$, a contradiction. ■

COROLLARY 49. — *$\mathcal{C}^+ \neq_{\mathrm{PFM}} \mathcal{PT}$ holds for all classes $\mathcal{C} \in \{\mathcal{A}, \mathcal{U}, \mathcal{B}, \mathcal{P}\}$.*

To conclude, we have located the exact position of $\mathcal{PT}$ in EPH (recall Figure 1).

5. Yet another characterization of stability

In spite of the negative results presented in the previous section, our further objective is to pursue non-modular alternatives in Section 6. To enable a particular translation technique therein, we concentrate here on establishing a new *characterization* of stable models based on supported models possessing a *level numbering* $\#$. Roughly speaking, the idea is to extend level numbers, first introduced for positive programs and their least models in Section 2.2, to the case of normal programs.

As suggested by the definition of supported models, we define the set of *supporting rules* $\mathrm{SR}(P, I) = \{r \in P \mid I \models \mathrm{B}(r)\} \subseteq P$ for any normal program P and an interpretation $I \subseteq \mathrm{Hb}(P)$. Thus each rule $r \in \mathrm{SR}(P, I)$ provides a support for $\mathrm{H}(r)$.

DEFINITION 50. — *Let M be a supported model of a normal program P. A function $\#$ from $M \cup \mathrm{SR}(P, M)$ to $\mathbb{N}$ is a level numbering w.r.t. M iff for every atom $\mathsf{a} \in M$:*

$$\#\mathsf{a} = \min\{\#r \mid r \in \mathrm{SR}(P, M) \text{ and } \mathsf{a} = \mathrm{H}(r)\} \tag{15}$$

and for every rule $r \in \mathrm{SR}(P, M)$,

$$\#r = \begin{cases} \max\{\#\mathsf{b} \mid \mathsf{b} \in \mathrm{B}^+(r)\} + 1, & \text{if } \mathrm{B}^+(r) \neq \emptyset. \\ 1, & \text{otherwise.} \end{cases} \tag{16}$$

It is important to realize that a level numbering need not exist for every supported model. This is demonstrated by the following example.

EXAMPLE 51. — Consider a logic program P consisting of two rules $r_1 = \mathsf{a} \leftarrow \mathsf{b}$ and $r_2 = \mathsf{b} \leftarrow \mathsf{a}$. There are two supported models of P: $M_1 = \emptyset$ and $M_2 = \{\mathsf{a}, \mathsf{b}\}$. The first model has a trivial level numbering with a domain $M_1 \cup \mathrm{SR}(P, M_1) = \emptyset$. For M_2, the domain $M_2 \cup \mathrm{SR}(P, M_2) = M_2 \cup P$. The requirements in Definition 50 lead to four equations: $\#\mathsf{a} = \#r_1$, $\#r_1 = \#\mathsf{b} + 1$, $\#\mathsf{b} = \#r_2$, and $\#r_2 = \#\mathsf{a} + 1$. From these, we obtain $\#\mathsf{a} = \#\mathsf{a} + 2$. So there is no level numbering w.r.t. M_2. □

PROPOSITION 52. — *Let M be a supported model of a normal program P. If there is a level numbering $\#$ w.r.t. M, then $\#$ is unique.*

PROOF. — Let $\#_1$ and $\#_2$ be two level numberings w.r.t. M. It can be shown by induction on $\#_1(x) > 0$ that $\#_1(x) = \#_2(x)$ for every $x \in M \cup \mathrm{SR}(P, M)$. ■

An obvious question is how one can determine level numberings in practice. In fact, the scheme introduced in Section 2.2 can be extended to cover rules as well.

DEFINITION 53. — *Let P be a positive program and $M = \mathrm{LM}(P)$. Let us define level numbers $\mathrm{lev}(\mathsf{a})$ for atoms $\mathsf{a} \in M$ as in Section 2.2. Given any rule $r \in P$ such that $\mathrm{B}^+(r) = \mathrm{B}(r) \subseteq M$, define the level number*

$$\mathrm{lev}(r) = \begin{cases} \max\{\mathrm{lev}(\mathsf{b}) \mid \mathsf{b} \in \mathrm{B}^+(r)\} + 1, & \text{if } \mathrm{B}^+(r) \neq \emptyset. \\ 1, & \text{otherwise.} \end{cases} \tag{17}$$

Assigning level numbers in this way is compatible with Definition 50.

LEMMA 54 ([JAN 03B]). — *Let P be a positive program, $M = \mathrm{LM}(P)$, and $\mathsf{a} \in M$.*

1) For every $r \in \mathrm{SR}(P, M)$ such that $\mathrm{H}(r) = \mathsf{a}$, $\mathrm{lev}(r) \geq \mathrm{lev}(\mathsf{a})$.

2) There is $r \in \mathrm{SR}(P, M)$ such that $\mathrm{H}(r) = \mathsf{a}$ and $\mathrm{lev}(r) = \mathrm{lev}(\mathsf{a})$.

Then the characterization of stable models is then established as follows.

THEOREM 55. — *Let P be a normal program.*

1) If M is a stable model of P, then M is a supported model of P and there exists a unique level numbering $\# : M \cup \mathrm{SR}(P, M) \to \mathbb{N}$ w.r.t. M defined as follows.

(i) For $\mathsf{a} \in M$, let $\#\mathsf{a} = \mathrm{lev}(\mathsf{a})$.

(ii) For $r \in \mathrm{SR}(P, M)$, let $\#r = \mathrm{lev}(r^+)$.

2) If M is a supported model of P and there is a level numbering $\#$ w.r.t. M, then $\#$ is unique and M is a stable model of P.

PROOF. — (1) Let M be a stable model of P. Then M is also a supported model of P [MAR 92]. Recall that each $r \in \mathrm{SR}(P, M)$ satisfies $M \models \mathrm{B}(r)$ which implies that $r^+ \in P^M$, $\mathrm{B}^+(r) \subseteq M$, and $\mathrm{H}(r) \in M$, as M is also a classical model of P. Let us now establish the requirements of Definition 50.

Consider any $\mathsf{a} \in M$. It should be established that $\#\mathsf{a}$ is the minimum among $\{\#r \mid r \in \mathrm{SR}(P, M) \text{ and } \mathrm{H}(r) = \mathsf{a}\}$. It is clear that this set is non-empty, as M is a supported model of P. Then consider any $r \in \mathrm{SR}(P, M)$ such that $\mathrm{H}(r) = \mathsf{a}$. Now $\#r$ is defined as $\mathrm{lev}(r^+)$ given in (17). Since $r \in \mathrm{SR}(P, M)$, we obtain $r^+ \in \mathrm{SR}(P^M, M)$. Thus $\#r = \mathrm{lev}(r^+) \geq \mathrm{lev}(\mathsf{a})$ by the first claim of Lemma 54. By the second claim, there is a rule $r' \in \mathrm{SR}(P^M, M)$ such that $\mathrm{H}(r') = \mathsf{a}$ and $\mathrm{lev}(r') = \mathrm{lev}(\mathsf{a})$. Then there is a rule $r'' \in \mathrm{SR}(P, M)$ such that $r' = (r'')^+$, $\mathrm{H}(r'') = \mathsf{a}$ and $\#r'' = \mathrm{lev}(r'') = \mathrm{lev}(\mathsf{a})$. Thus $\#\mathsf{a} = \mathrm{lev}(\mathsf{a})$ is the minimum in question. Then consider any $r \in \mathrm{SR}(P, M)$. There are two possibilities. If $\mathrm{B}^+(r) = \emptyset$, we obtain $\mathrm{B}^+(r^+) = \emptyset$ so that $\#r = \mathrm{lev}(r^+) = 1$ by (17). This is in perfect harmony with Definition 50. On the other hand, if $\mathrm{B}^+(r) \neq \emptyset$, then $\#r = \mathrm{lev}(r^+)$ is defined as $\max\{\mathrm{lev}(\mathsf{b}) \mid \mathsf{b} \in \mathrm{B}^+(r^+)\} + 1$. Since $\mathrm{B}^+(r^+) = \mathrm{B}^+(r)$ and $\#\mathsf{b} = \mathrm{lev}(\mathsf{b})$ for each $\mathsf{b} \in \mathrm{B}^+(r)$ by definition, we know that $\#r = \mathrm{lev}(r^+)$ coincides with $\max\{\#\mathsf{b} \mid \mathsf{b} \in \mathrm{B}^+(r)\} + 1$ as insisted by Definition 50. Thus $\#$ is a level numbering w.r.t. M and the uniqueness of $\#$ follows by Proposition 52.

(2) Let M be a supported model of P and $\#$ a level numbering w.r.t. M. The uniqueness of $\#$ follows by Proposition 52. It follows that $M \models P$ and $M \models P^M$. Thus it is immediately clear that $\mathrm{LM}(P^M)$ is contained in M. It remains to prove that $M \subseteq \mathrm{LM}(P^M)$. We use complete induction on $\#\mathsf{a} > 1$ to show that $\mathsf{a} \in M$ implies $\mathsf{a} \in \mathrm{LM}(P^M)$. **Base case:** $\#\mathsf{a} = 1$. Suppose that $\mathsf{a} \in M$. Since $\#\mathsf{a} = 1$, the only possibility is that there is $r \in \mathrm{SR}(P, M)$ with $\mathrm{H}(r) = \mathsf{a}$ and $\mathrm{B}^+(r) = \emptyset$. It follows that $\mathsf{a} \leftarrow$ belongs to P^M so that $\mathsf{a} \in \mathrm{LM}(P^M)$. **Induction step:** $\#\mathsf{a} = n > 1$. Suppose that $\mathsf{a} \in M$. Since $\#\mathsf{a} > 1$, there is a rule $r \in \mathrm{SR}(P, M)$ such

that $\mathrm{H}(r) = \mathsf{a}$, $\mathrm{B}^+(r) \neq \emptyset$, and $\#\mathsf{a} = \#r$. Then consider any $\mathsf{b} \in \mathrm{B}^+(r)$. Since $\#r = \max\{\#\mathsf{b}' \mid \mathsf{b}' \in \mathrm{B}^+(r)\} + 1$, we obtain $\#\mathsf{b} < n$. Moreover $r \in \mathrm{SR}(P, M)$ implies that $\mathsf{b} \in M$. Thus $\mathsf{b} \in \mathrm{LM}(P^M)$ by the inductive hypothesis and we have established that $\mathrm{B}^+(r) \subseteq \mathrm{LM}(P^M)$. On the other hand, $r \in \mathrm{SR}(P, M)$ implies that $r^+ = \mathsf{a} \leftarrow \mathrm{B}^+(r) \in P^M$. It follows that $\mathsf{a} \in \mathrm{LM}(P^M)$. ■

6. Non-modular translation functions

In Section 4, we show that faithful and modular translations cannot be established between certain classes of logic programs. However, this does not exclude the possibility that a polynomial and faithful, but *non-modular* translation function could be devised for the classes involved. Such alternatives are taken into consideration now. We proceed as follows. Section 6.1 covers the case of positive programs for which non-modular alternatives are easy to obtain. A faithful and non-modular translation function from normal programs to atomic normal programs is developed in Section 6.2. This is a far more complicated objective as the number of stable models may vary. Finally, we conduct a comparison with propositional theories in Section 6.3.

6.1. *Positive programs revisited*

Theorem 39 states that $\mathcal{B}^+ \not\leq_{\mathrm{FM}} \mathcal{U}^+$ and $\mathcal{U}^+ \not\leq_{\mathrm{FM}} \mathcal{A}^+$. In spite of these relationships, it is straightforward to obtain a faithful and non-modular translation in case of positive programs. Basically, this boils down to the fact that the least model $\mathrm{LM}(P)$ can be be computed in polynomial time for any $P \in \mathcal{P}^+$.

DEFINITION 56. — *For any $P \in \mathcal{P}^+$, define $\mathrm{Tr}_{\mathrm{LM}}(P) = \{\mathsf{a} \leftarrow \mid \mathsf{a} \in \mathrm{LM}(P)\}$. Moreover, let $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{LM}}(P)) = \mathrm{Hb}_{\mathrm{v}}(P)$ and $\mathrm{Hb}_{\mathrm{h}}(\mathrm{Tr}_{\mathrm{LM}}(P)) = \mathrm{Hb}_{\mathrm{h}}(P)$ so that $\mathrm{Hb}_{\mathrm{a}}(\mathrm{Tr}_{\mathrm{LM}}(P)) = \mathrm{Hb}(P) - \mathrm{LM}(P)$.*

THEOREM 57. — $\mathcal{P}^+ \leq_{\mathrm{PF}} \mathcal{A}^+$.

PROOF. — It is clear that $\mathrm{Tr}_{\mathrm{LM}}$ is faithful, since $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{LM}}(P)) = \mathrm{Hb}_{\mathrm{v}}(P)$ by definition and Lemma 25 implies that both P and $\mathrm{Tr}_{\mathrm{LM}}(P)$ have a unique stable model $\mathrm{LM}(P) = \mathrm{LM}(\mathrm{Tr}_{\mathrm{AT}}(P))$. Moreover, $\mathrm{Tr}_{\mathrm{LM}}$ is polynomial, as $\mathrm{LM}(P)$ can be computed in polynomial time. The iterative characterization from Section 2.2 leads to a quadratic algorithm, but there is also a linear time algorithm [DOW 84]. ■

COROLLARY 58. — $\mathcal{A}^+ =_{\mathrm{PF}} \mathcal{U}^+ =_{\mathrm{PF}} \mathcal{B}^+ =_{\mathrm{PF}} \mathcal{P}^+$.

The relations $\leq_{\mathrm{PFM}}$ and $\leq_{\mathrm{PF}}$ give rise to diverse classifications for the classes of positive logic programs. In fact, the relation $\leq_{\mathrm{PFM}}$ is more accurate so that the hierarchy obtained with $\leq_{\mathrm{PFM}}$ is more refined than the one obtained with $\leq_{\mathrm{PF}}$.

EXAMPLE 59. — The programs B, A_1, A_2 and A_3 from the the proof of Theorem 32 are translated as follows: $\mathrm{Tr}_{\mathrm{LM}}(B \cup A_1 \cup A_2) = \mathrm{Tr}_{\mathrm{LM}}(B \cup A_2 \cup A_3) = \mathrm{Tr}_{\mathrm{LM}}(B \cup A_3 \cup A_1) = \{\mathsf{a} \leftarrow;\ \mathsf{b} \leftarrow;\ \mathsf{c} \leftarrow\}$, but $\mathrm{Tr}_{\mathrm{LM}}(B) = \emptyset$ with $\mathrm{Hb}_{\mathrm{a}}(\mathrm{Tr}_{\mathrm{LM}}(B)) = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$,

$\mathrm{Tr}_{\mathrm{LM}}(A_1) = \{\mathsf{a} \leftarrow\}$, $\mathrm{Tr}_{\mathrm{LM}}(A_2) = \{\mathsf{b} \leftarrow\}$, and $\mathrm{Tr}_{\mathrm{LM}}(A_3) = \{\mathsf{c} \leftarrow\}$. Thus $\mathrm{Tr}_{\mathrm{LM}}$ is clearly non-modular: $\mathrm{Tr}_{\mathrm{LM}}(B \cup A_1 \cup A_2) \neq \mathrm{Tr}_{\mathrm{LM}}(B) \cup \mathrm{Tr}_{\mathrm{LM}}(A_1) \cup \mathrm{Tr}_{\mathrm{LM}}(A_2)$. □

Generally speaking, a translation obtained with a *non-modular* translation function $\mathrm{Tr}_{\mathrm{NM}}$ is often heavily dependent on the program P being translated. Already slight changes to P may alter $\mathrm{Tr}_{\mathrm{NM}}(P)$ completely. This reveals one shortcoming of non-modular translations: they do not support updates very well. E.g., in Example 59, the effect of removing A_2 from $\mathrm{Tr}_{\mathrm{LM}}(B \cup A_1 \cup A_2)$ is drastic as $\mathrm{Tr}_{\mathrm{LM}}(B \cup A_1) = \emptyset$.

6.2. *Translating normal programs into atomic ones*

As shown in Section 6.1, it is easy to obtain a non-modular and faithful translation function for removing positive body literals in the case of positive programs. The setting becomes far more complicated when normal logic programs are taken into consideration, since a normal program may possess several stable models and it is not clear how to apply $\mathrm{Tr}_{\mathrm{LM}}$ from Definition 56. Nevertheless, we intend to develop a polynomial and faithful translation function $\mathrm{Tr}_{\mathrm{AT}}$ so that an arbitrary normal program P gets translated into an atomic program $\mathrm{Tr}_{\mathrm{AT}}(P)$. It is clear by the results presented in Section 4.2 that $\mathrm{Tr}_{\mathrm{AT}}$ must be non-modular if faithfulness is to be expected. Our idea is to apply the characterization of stable models developed in Section 5 so that each stable model M of a normal program P is eventually captured as a supported model M of P possessing a level numbering w.r.t. M. To recall the basic concepts from Section 5 we give an example of a level numbering.

EXAMPLE 60. — Let $P = \{r_1, r_2, r_3\}$ be a (positive) normal program consisting of the rules $r_1 = \mathsf{a} \leftarrow$; $r_2 = \mathsf{a} \leftarrow \mathsf{b}$; and $r_3 = \mathsf{b} \leftarrow \mathsf{a}$ so that $\mathrm{Hb}_{\mathrm{v}}(P) = \mathrm{Hb}(P) = \{\mathsf{a}, \mathsf{b}\}$. The unique stable model $M = \mathrm{LM}(P) = \{\mathsf{a}, \mathsf{b}\}$ is supported by $\mathrm{SR}(P, M) = P$. The unique level numbering $\#$ w.r.t. M is determined by $\#r_1 = 1$, $\#\mathsf{a} = 1$, $\#r_3 = 2$, $\#\mathsf{b} = 2$, and $\#r_2 = 3$. □

However, there is no explicit way of representing a level numbering within a normal program and we have to encode such a numbering using propositional atoms. Then a natural solution is to use a binary representation for actual level numbers. In the worst case, every atom in $\mathrm{Hb}(P)$ is assigned a different level number as demonstrated in Example 60. Thus the level numbers of atoms may vary from 1 to $|\mathrm{Hb}(P)|$. Hence the highest possible level number of a rule $r \in P$ is $|\mathrm{Hb}(P)| + 1$, as for r_2 in our example. By leaving room for 0 as the least binary value, we have to be prepared for binary numbers consisting of at most $\nabla P = \lceil \log_2(|\mathrm{Hb}(P)| + 2) \rceil$ bits.

PROPOSITION 61 ([JAN 03B]). — *If M is a supported model of a normal program P and $\# : M \cup \mathrm{SR}(P, M) \rightarrow \mathbb{N}$ is a level numbering w.r.t. M, then $0 < \#\mathsf{a} < 2^{\nabla P} - 1$ for every $\mathsf{a} \in M$ and $0 < \#r < 2^{\nabla P}$ for every $r \in \mathrm{SR}(P, M)$.*

The logarithmic factor embodied in ∇P forms an important design criterion for us in order to keep the length of the translation $||\mathrm{Tr}_{\mathrm{AT}}(P)||$ as well as the translation time proportional to $||P|| \times \nabla P$ rather than $||P|| \times |\mathrm{Hb}(P)|$. Hence we strive for a

sub-quadratic translation function from $\mathcal{P}$ to $\mathcal{A}$. To get an idea of the potential behind such an objective, we have $\nabla P = 14$ for a program P with $|\mathrm{Hb}(P)| = 10000$.

6.2.1. *Representing binary counters*

We have to fix some notation in order to deal with binary representations of natural numbers. Given the number of bits b and a natural number $0 \leq n < 2^b$, we write $n[i \ldots j]$, where $0 < i \leq j \leq b$, for the binary representation of n from the i^{th} bit to the j^{th} bit in the decreasing order of significance. Thus $n[1 \ldots b]$ gives a complete binary representation for n. Moreover, as a special case of this notation, we may refer to the i^{th} bit simply by writing $n[i] = n[i \ldots i]$.

Technically speaking, the idea is to encode the level number $\#\mathsf{a}$ for a particular atom $\mathsf{a} \in \mathrm{Hb}(P)$ using a *vector* $\mathsf{a}_1, \ldots, \mathsf{a}_j$ of new atoms where $j = \nabla P$. Such a vector can be understood as a representation of a *binary counter* of j bits; the first and the last atoms corresponding to the most significant and the least significant bits, respectively. The idea is to equate bits 0 and 1 with the truth values false and true assigned to atoms, since atoms may take only two values under the stable model semantics. Because the resulting translation is supposed to be an atomic normal program, positive body literals are forbidden and we have to introduce the vector $\overline{\mathsf{a}_1}, \ldots, \overline{\mathsf{a}_j}$ of complementary atoms so that we can condition rules on both values of bits. This is exactly the technique that was demonstrated in Example 34. In the current setting, the idea is that the i^{th} bit of the binary counter associated with the atom a takes the value 0 (resp. 1) if and only if a_i (resp. $\overline{\mathsf{a}_i}$) cannot be inferred, i.e. the negative literal ${\sim}\mathsf{a}_i$ (resp. ${\sim}\overline{\mathsf{a}_i}$) is satisfied in rule bodies. In the sequel, we may introduce a binary counter of the kind above for any atom a by subscripting it with an index i in the range $0 < i \leq j$.

In order to express the constraints on level numberings, as demanded by Definition 50, we need certain primitive operations on binary counters. The respective subprograms are listed in Table 2. The size of each subprogram is governed by a parameter j which gives the number of bits used in the binary counters involved. The activation of all subprograms is controlled by an additional atom c. The idea is that the respective subprograms are activated only when c cannot be inferred, i.e. c is false under stable model semantics. Whenever c is true, all atoms involved in these subprograms are false by default. In the following, we give brief descriptions of the primitives.

1) The subprogram $\mathrm{SEL}_j(\mathsf{a}, \mathsf{c})$ selects a value between 0 and $2^j - 1$ for the binary counter $\mathsf{a}_1, \ldots, \mathsf{a}_j$ associated with an atom a.

2) The program $\mathrm{NXT}_j(\mathsf{a}, \mathsf{b}, \mathsf{c})$ binds the values of the binary counters associated with atoms a and b, respectively, so that the latter is the former increased by one (mod 2^j). Proposition 61 tells us that ∇P is big enough to prevent counters from wrapping. The design of $\mathrm{NXT}_j(\mathsf{a}, \mathsf{b}, \mathsf{c})$ is economical as it does not involve explicit carry bits and it is based on the following observations about increasing a binary counter by one (mod 2^j). First, the least significant bit, i.e. the j^{th} one, is always changed; either from 0 to 1 or from 1 to 0. Second, any other bit (say i^{th} where $1 \leq i < j$) is changed only if the next less significant bit, i.e. the $i + 1^{\text{th}}$ one, changes from 1 to 0. To see this, it may be instructive to study the transition from 010111 to 011000 when $j = 6$.

Table 2. *Encoding primitive operations for binary counters*

Primitive	Definition with atomic rules
$\mathrm{SEL}_j(\mathsf{a},\mathsf{c})$	$\{\mathsf{a}_i \leftarrow {\sim}\overline{\mathsf{a}_i}, {\sim}\mathsf{c};\ \ \overline{\mathsf{a}_i} \leftarrow {\sim}\mathsf{a}_i, {\sim}\mathsf{c} \mid 0 < i \leq j\}$
$\mathrm{NXT}_j(\mathsf{a},\mathsf{b},\mathsf{c})$	$\{\mathsf{b}_i \leftarrow {\sim}\mathsf{a}_i, {\sim}\overline{\mathsf{a}_{i+1}}, {\sim}\mathsf{b}_{i+1}, {\sim}\mathsf{c} \mid 0 < i < j\} \cup$
	$\{\mathsf{b}_i \leftarrow {\sim}\overline{\mathsf{a}_i}, {\sim}\mathsf{a}_{i+1}, {\sim}\mathsf{c} \mid 0 < i < j\} \cup$
	$\{\mathsf{b}_i \leftarrow {\sim}\overline{\mathsf{a}_i}, {\sim}\overline{\mathsf{b}_{i+1}}, {\sim}\mathsf{c} \mid 0 < i < j\} \cup$
	$\{\overline{\mathsf{b}_i} \leftarrow {\sim}\mathsf{b}_i, {\sim}\mathsf{c} \mid 0 < i < j\} \cup$
	$\{\overline{\mathsf{b}_j} \leftarrow {\sim}\overline{\mathsf{a}_j}, {\sim}\mathsf{c};\ \ \mathsf{b}_j \leftarrow {\sim}\mathsf{a}_j, {\sim}\mathsf{c}\}$
$\mathrm{FIX}_j(\mathsf{a},n,\mathsf{c})$	$\{\overline{\mathsf{a}_i} \leftarrow {\sim}\mathsf{c} \mid 0 < i \leq j \text{ and } n[i] = 0\} \cup$
	$\{\mathsf{a}_i \leftarrow {\sim}\mathsf{c} \mid 0 < i \leq j \text{ and } n[i] = 1\}$
$\mathrm{LT}_j(\mathsf{a},\mathsf{b},\mathsf{c})$	$\{\mathsf{lt}(\mathsf{a},\mathsf{b})_i \leftarrow {\sim}\mathsf{a}_i, {\sim}\overline{\mathsf{b}_i}, {\sim}\mathsf{c} \mid 0 < i \leq j\} \cup$
	$\{\mathsf{lt}(\mathsf{a},\mathsf{b})_i \leftarrow {\sim}\mathsf{a}_i, {\sim}\mathsf{b}_i, {\sim}\overline{\mathsf{lt}(\mathsf{a},\mathsf{b})_{i+1}}, {\sim}\mathsf{c} \mid 0 < i < j\} \cup$
	$\{\mathsf{lt}(\mathsf{a},\mathsf{b})_i \leftarrow {\sim}\overline{\mathsf{a}_i}, {\sim}\overline{\mathsf{b}_i}, {\sim}\overline{\mathsf{lt}(\mathsf{a},\mathsf{b})_{i+1}}, {\sim}\mathsf{c} \mid 0 < i < j\} \cup$
	$\{\overline{\mathsf{lt}(\mathsf{a},\mathsf{b})_i} \leftarrow {\sim}\mathsf{lt}(\mathsf{a},\mathsf{b})_i, {\sim}\mathsf{c} \mid 0 < i \leq j\}$
$\mathrm{EQ}_j(\mathsf{a},\mathsf{b},\mathsf{c})$	$\{\overline{\mathsf{eq}(\mathsf{a},\mathsf{b})} \leftarrow {\sim}\mathsf{a}_i, {\sim}\overline{\mathsf{b}_i}, {\sim}\mathsf{c} \mid 0 < i \leq j\} \cup$
	$\{\overline{\mathsf{eq}(\mathsf{a},\mathsf{b})} \leftarrow {\sim}\overline{\mathsf{a}_i}, {\sim}\mathsf{b}_i, {\sim}\mathsf{c} \mid 0 < i \leq j\} \cup$
	$\{\mathsf{eq}(\mathsf{a},\mathsf{b}) \leftarrow {\sim}\overline{\mathsf{eq}(\mathsf{a},\mathsf{b})}, {\sim}\mathsf{c}\}$

3) The subprogram $\mathrm{FIX}_j(\mathsf{a},n,\mathsf{c})$ assigns a fixed value $0 \leq n < 2^j$, in the binary representation, to the counter associated with the atom a.

4) The program $\mathrm{LT}_j(\mathsf{a},\mathsf{b},\mathsf{c})$ checks if the value of the binary counter associated with an atom a is strictly lower than the value of the binary counter associated with another atom b. To keep the program linear in j, we need a vector of new atoms $\mathsf{lt}(\mathsf{a},\mathsf{b})_1,\ldots,\mathsf{lt}(\mathsf{a},\mathsf{b})_j$ plus the corresponding vector of complementary atoms. The atoms $\mathsf{lt}(\mathsf{a},\mathsf{b})_1$ and $\overline{\mathsf{lt}(\mathsf{a},\mathsf{b})_1}$, which refer to the most significant bits, capture the result.

5) The subprogram $\mathrm{EQ}_j(\mathsf{a},\mathsf{b},\mathsf{c})$ checks if the counters associated with the atoms a and b hold the same value. The new atoms $\mathsf{eq}(\mathsf{a},\mathsf{b})$ and $\overline{\mathsf{eq}(\mathsf{a},\mathsf{b})}$ capture the result.

Our next goal is to specify the intended outcomes of the primitives listed in Table 2. Whenever the value of a counter of j bits associated with an atom a is chosen to be $0 \leq n < 2^j$, the contribution of the respective program $\mathrm{SEL}_j(\mathsf{a},\mathsf{c})$ is a set of atoms $\mathrm{AT}_j^{\mathrm{ctr}}(\mathsf{a},n) = \{\mathsf{a}_i \mid 0 < i \leq j \text{ and } n[i] = 1\} \cup \{\overline{\mathsf{a}_i} \mid 0 < i \leq j \text{ and } n[i] = 0\}$ given that c is not inferable. The subprogram $\mathrm{NXT}_j(\mathsf{b},\mathsf{a},\mathsf{c})$ is supposed to produce the same set of atoms $\mathrm{AT}_j^{\mathrm{ctr}}(\mathsf{a},n)$ given that the counter associated with some other atom b is holding a value m such that $n = m + 1 \bmod 2^j$. On the other hand, the

result of a subprogram $\mathrm{LT}_j(\mathsf{a},\mathsf{b},\mathsf{c})$, when the atom c is assigned to false, is given in (18) below. It is assumed that the values of the counters associated with the atoms a and b are n and m in the ranges $0 \leq n < 2^j$ and $0 \leq m < 2^j$, respectively.

$$\mathrm{AT}^{\mathrm{lt}}_j(\mathsf{a},n,\mathsf{b},m) = \{\mathsf{lt}(\mathsf{a},\mathsf{b})_i \mid 0 < i \leq j \text{ and } n[i\ldots j] < m[i\ldots j]\} \cup \{\overline{\mathsf{lt}(\mathsf{a},\mathsf{b})_i} \mid 0 < i \leq j \text{ and } n[i\ldots j] \geq m[i\ldots j]\}. \quad (18)$$

The result of testing the equality of the counters is defined analogously. The outcome for $\mathrm{EQ}_j(\mathsf{a},\mathsf{b},\mathsf{c})$, when c is not inferable, is $\mathrm{AT}^{\mathrm{eq}}_j(\mathsf{a},n,\mathsf{b},m) = \{\mathsf{eq}(\mathsf{a},\mathsf{b}) \mid n = m\} \cup \{\overline{\mathsf{eq}(\mathsf{a},\mathsf{b})} \mid n \neq m\}$. Finally, given an atom a and a set of atoms N — such as a stable model of a program involving the subprograms under consideration — we may *extract* the value of the counter $\mathsf{a}_1,\ldots,\mathsf{a}_j$ associated with an atom a by

$$\mathrm{val}_j(\mathsf{a},N) = \sum\{2^{j-i} \mid 0 < i \leq j \text{ and } \mathsf{a}_i \in N\}. \quad (19)$$

It follows that $\mathrm{val}_j(\mathsf{a},N)[i] = 1 \iff \mathsf{a}_i \in N$ holds for each $0 < i \leq j$. Moreover, we have $\mathrm{val}_j(\mathsf{a},\mathrm{AT}^{\mathrm{ctr}}_j(\mathsf{a},n)) = n$ for any $0 \leq n < 2^j$.

6.2.2. *A non-modular translation function*

In this section, we concentrate on composing a non-modular translation function $\mathrm{Tr}_{\mathrm{AT}} : \mathcal{P} \to \mathcal{A}$ in four steps by applying the characterization of stable models from Section 5. Accordingly, the overall translation $\mathrm{Tr}_{\mathrm{AT}}(P)$ of a normal logic program P will divide in four parts summarized below:

1) $\mathrm{Tr}_{\mathrm{SUPP}}(P)$ that captures a supported model M of P;

2) $\mathrm{Tr}_{\mathrm{CTR}}(P)$ that represents a level numbering candidate $\#$ in terms of counters;

3) $\mathrm{Tr}_{\mathrm{MIN}}(P)$ that imposes constraints on the candidate $\#$ and its domain $M \cup \mathrm{SR}(P,M)$ so that each atom $\mathsf{a} \in M$ satisfies (15); and

4) $\mathrm{Tr}_{\mathrm{MAX}}(P)$ that similarly enforces (16) for each rule $r \in \mathrm{SR}(P,M)$.

The details of these translations will be given in Definitions 62–65 to be presented below. Of course, we aim at a faithful translation $\mathrm{Tr}_{\mathrm{AT}}(P)$ that captures each stable model $M \in \mathrm{SM}(P)$ with some stable model $N \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{AT}}(P))$ in a bijective fashion. In the subsequent informal discussion, we use M and N as a pair of stable models involved in the resulting bijective relationship. However, the description of the exact structure of N is postponed until Section 6.2.3 where the polynomiality and faithfulness of $\mathrm{Tr}_{\mathrm{AT}}$ is eventually established in Theorem 74. Meanwhile, Example 67 may be useful for the reader to better access the forthcoming definition of $\mathrm{Tr}_{\mathrm{AT}}$.

The first part of the translation $\mathrm{Tr}_{\mathrm{AT}}(P)$, i.e. $\mathrm{Tr}_{\mathrm{SUPP}}(P)$, defines the complementary atom $\overline{\mathsf{a}}$ for each atom $\mathsf{a} \in \mathrm{Hb}(P)$. Such atoms enable the removal of positive subgoals using the technique already presented in Example 34, i.e. $\mathsf{a} \in \mathrm{B}^+(r)$ in a rule r is roughly captured by a negative literal $\sim\overline{\mathsf{a}}$. However, spurious (supported) models will result and we need the rest of $\mathrm{Tr}_{\mathrm{AT}}(P)$ to exclude the non-stable ones.

DEFINITION 62. — *For a normal program P, define an atomic normal program*

$$\begin{aligned}\mathrm{Tr}_{\mathrm{SUPP}}(P) = \{\overline{\mathsf{a}} \leftarrow {\sim}\mathsf{a} \mid \mathsf{a} \in \mathrm{Hb}(P)\} \cup \\ \{\mathrm{H}(r) \leftarrow {\sim}\overline{\mathsf{bt}(r)};\ \overline{\mathsf{bt}(r)} \leftarrow {\sim}\mathsf{bt}(r) \mid r \in P\} \cup \\ \{\mathsf{bt}(r) \leftarrow {\sim}\overline{\mathrm{B}^+(r)}, {\sim}\mathrm{B}^-(r) \mid r \in P\}.\end{aligned} \tag{20}$$

Basically, a rule $r \in P$ could be rewritten as $\mathrm{H}(r) \leftarrow {\sim}\overline{\mathrm{B}^+(r)}, {\sim}\mathrm{B}^-(r)$, but other parts of the overall translation require us to determine when the *body* of r is *true*. This is why new atoms $\mathsf{bt}(r)$ and $\overline{\mathsf{bt}(r)}$ are introduced for each $r \in P$. Note that copying the transformed body of r to other parts of the translation would imply a quadratic blow-up and we need $\mathsf{bt}(r)$ for each $r \in P$ in order to save space. The next part of the translation introduces binary counters which represent a level numbering candidate.

DEFINITION 63. — *For a normal program P, define an atomic normal program*

$$\begin{aligned}\mathrm{Tr}_{\mathrm{CTR}}(P) = \bigcup_{\mathsf{a} \in \mathrm{Hb}(P)} [\mathrm{SEL}_{\nabla P}(\mathsf{ctr}(\mathsf{a}), \overline{\mathsf{a}}) \cup \mathrm{NXT}_{\nabla P}(\mathsf{ctr}(\mathsf{a}), \mathsf{nxt}(\mathsf{a}), \overline{\mathsf{a}})] \cup \\ \bigcup_{r \in P \text{ and } \mathrm{B}^+(r) = \emptyset} \mathrm{FIX}_{\nabla P}(\mathsf{ctr}(r), 1, \overline{\mathsf{bt}(r)}) \cup \\ \bigcup_{r \in P \text{ and } \mathrm{B}^+(r) \neq \emptyset} \mathrm{SEL}_{\nabla P}(\mathsf{ctr}(r), \overline{\mathsf{bt}(r)}).\end{aligned} \tag{21}$$

In this way, two new atoms $\mathsf{ctr}(\mathsf{a})$ and $\mathsf{nxt}(\mathsf{a})$, which act as names of two counters, are introduced for each atom $\mathsf{a} \in \mathrm{Hb}(P)$. The eventual purpose of these counters is to hold the values $\#\mathsf{a}$ and $\#\mathsf{a} + 1$, respectively, in case that a belongs to the domain of a level numbering $\#$, i.e. $\mathsf{a} \in M$; or equivalently, $\overline{\mathsf{a}} \notin N$. However, at this point, the primitives included in $\mathrm{Tr}_{\mathrm{CTR}}(P)$ choose a value for $\mathsf{ctr}(\mathsf{a})$ and define the value of $\mathsf{nxt}(\mathsf{a})$ as the successor of the value of $\mathsf{ctr}(\mathsf{a})$ modulo $2^{\nabla P}$. Quite similarly, a new atom $\mathsf{ctr}(r)$ and the respective counter is introduced for each $r \in P$ to eventually hold $\#r$ when r is in the domain of $\#$, i.e. $r \in \mathrm{SR}(P, M)$, or equivalently $\overline{\mathsf{bt}(r)} \notin N$. In case of an atomic rule $r \in P$ with $\mathrm{B}^+(r) = \emptyset$, the counter $\mathsf{ctr}(r)$ is assigned a fixed value 1 and no choice is made in accordance with Definition 50.

The translation $\mathrm{Tr}_{\mathrm{CTR}}(P)$ is sufficient for choosing a candidate level numbering for a supported model M of P that is to be captured by the rules in $\mathrm{Tr}_{\mathrm{SUPP}}(P)$. We have to introduce constraints in order to ensure that the candidate is indeed a level numbering, as dictated by Definition 50. We start with the conditions imposed on rules $r \in P$ and in particular, when $r \in \mathrm{SR}(P, M)$ holds, i.e. $M \models \mathrm{B}(r)$. This explains why $\overline{\mathsf{bt}(r)}$ is used as a controlling atom[10] in the forthcoming translation. As explained above, the case of atomic rules $r \in P$ with $\mathrm{B}^+(r) = \emptyset$ is already covered by

10. Recall that such atoms appear as negative conditions in the subprograms listed in Table 2.

$\mathrm{Tr}_{\mathrm{CTR}}(P)$ which assigns a fixed value — the natural number 1 — to $\mathsf{ctr}(r)$. But for non-atomic rules $r \in P$ with $\mathrm{B}^+(r) \neq \emptyset$, the maximization principle from Definition 50 must be expressed e.g. as follows.

DEFINITION 64. — *Let* x *be a new atom not appearing in* $\mathrm{Hb}(P)$. *For an non-atomic rule* $r \in P$ *and a number of bits* b, *define* $\mathrm{Tr}_{\mathrm{MAX}}(r, b) = \bigcup_{\mathsf{a} \in \mathrm{B}^+(r)} \mathrm{Tr}_{\mathrm{MAX}}(r, b, \mathsf{a})$ *where for any* $\mathsf{a} \in \mathrm{B}^+(r)$, *the translation* $\mathrm{Tr}_{\mathrm{MAX}}(r, b, \mathsf{a}) =$

$$\begin{aligned}
&\mathrm{LT}_b(\mathsf{ctr}(r), \mathsf{nxt}(\mathsf{a}), \overline{\mathsf{bt}(r)}) \cup \mathrm{EQ}_b(\mathsf{ctr}(r), \mathsf{nxt}(\mathsf{a}), \overline{\mathsf{bt}(r)}) \cup \\
&\{\mathsf{x} \leftarrow {\sim}\mathsf{x}, {\sim}\overline{\mathsf{bt}(r)}, {\sim}\overline{\mathsf{lt}(\mathsf{ctr}(r), \mathsf{nxt}(\mathsf{a}))_1}\} \cup \\
&\{\mathsf{max}(r) \leftarrow {\sim}\overline{\mathsf{bt}(r)}, {\sim}\overline{\mathsf{eq}(\mathsf{ctr}(r), \mathsf{nxt}(\mathsf{a}))}\}.
\end{aligned}$$

For a normal program P, *define an atomic program* $\mathrm{Tr}_{\mathrm{MAX}}(P) = \bigcup\{\mathrm{Tr}_{\mathrm{MAX}}(r, \nabla P) \mid r \in P$ *and* $\mathrm{B}^+(r) \neq \emptyset\} \cup \{\mathsf{x} \leftarrow {\sim}\mathsf{x}, {\sim}\overline{\mathsf{bt}(r)}, {\sim}\mathsf{max}(r) \mid r \in P$ *and* $\mathrm{B}^+(r) \neq \emptyset\}$.

An informal description follows. The rules in $\mathrm{Tr}_{\mathrm{MAX}}(r, \nabla P, \mathsf{a})$ are to be activated for a non-atomic rule $r \in \mathrm{SR}(P, M)$ and a positive body atom $\mathsf{a} \in \mathrm{B}^+(r)$. As a consequence, the value held by $\mathsf{ctr}(r)$ must be greater than or equal to the value of $\mathsf{nxt}(\mathsf{a})$ which is supposed to be the value of $\mathsf{ctr}(\mathsf{a})$ increased by one. In addition to this, the rules for $\mathsf{max}(r)$ in $\mathrm{Tr}_{\mathrm{MAX}}(r, \nabla P, \mathsf{a})$ and $\mathrm{Tr}_{\mathrm{MAX}}(P)$ make the value of $\mathsf{ctr}(r)$ equal to the value of $\mathsf{nxt}(\mathsf{a})$ for some $\mathsf{a} \in \mathrm{B}^+(r)$. Thus the value of $\mathsf{ctr}(r)$ must be the maximum among the values of the counters $\mathsf{nxt}(\mathsf{a})$ associated with the positive body atoms $\mathsf{a} \in \mathrm{B}^+(r)$. This conforms to the definition of $\#r$ given in Definition 50.

Let us then turn our attention to atoms $\mathsf{a} \in \mathrm{Hb}(P)$ that are assigned to true in a supported model M of P satisfying $M = \mathrm{T}_{P^M}(M)$. Then there is a rule $r \in \mathrm{SR}(P, M)$ such that $\mathrm{H}(r) = \mathsf{a}$. Moreover, the level number $\#\mathsf{a}$ is defined as the minimum among the respective rules by Definition 50.

DEFINITION 65. — *Let* y *be a new atom not appearing in* $\mathrm{Hb}(P)$. *For a rule* r *and a number of bits* b, *define* $\mathrm{Tr}_{\mathrm{MIN}}(r, b) =$

$$\begin{aligned}
&\mathrm{LT}_b(\mathsf{ctr}(r), \mathsf{ctr}(\mathrm{H}(r)), \overline{\mathsf{bt}(r)}) \cup \mathrm{EQ}_b(\mathsf{ctr}(r), \mathsf{ctr}(\mathrm{H}(r)), \overline{\mathsf{bt}(r)}) \cup \\
&\{\mathsf{y} \leftarrow {\sim}\mathsf{y}, {\sim}\overline{\mathsf{bt}(r)}, {\sim}\overline{\mathsf{lt}(\mathsf{ctr}(r), \mathsf{ctr}(\mathrm{H}(r)))_1}\} \cup \\
&\{\mathsf{min}(\mathrm{H}(r)) \leftarrow {\sim}\overline{\mathsf{bt}(r)}, {\sim}\overline{\mathsf{eq}(\mathsf{ctr}(r), \mathsf{ctr}(\mathrm{H}(r)))}\}.
\end{aligned}$$

For a normal program P, *define an atomic normal program*

$$\mathrm{Tr}_{\mathrm{MIN}}(P) = \bigcup_{r \in P} \mathrm{Tr}_{\mathrm{MIN}}(r, \nabla P) \cup \{\mathsf{y} \leftarrow {\sim}\mathsf{y}, {\sim}\overline{\mathsf{a}}, {\sim}\mathsf{min}(\mathsf{a}) \mid \mathsf{a} \in \mathrm{Hb}(P)\}. \quad (22)$$

Given $\mathsf{a} \in M$ and a rule $r \in \mathrm{SR}(P, M)$ such that $\mathrm{H}(r) = \mathsf{a}$, the rules in $\mathrm{Tr}_{\mathrm{MIN}}(r, \nabla P)$ make the value of $\mathsf{ctr}(\mathsf{a})$ lower than or equal to the value of $\mathsf{ctr}(r)$. Moreover, the rules for $\mathsf{min}(\mathsf{a})$ in $\mathrm{Tr}_{\mathrm{MIN}}(P)$ ensure that the value of $\mathsf{ctr}(\mathsf{a})$ equals to the value of $\mathsf{ctr}(r)$ for at least one such rule r. In this way, the value of $\mathsf{ctr}(\mathsf{a})$ becomes

necessarily the minimum, which is in harmony with the definition of $\#\mathsf{a}$ in Definition 50. We are now ready to combine the four translations presented in Definitions 62–65.

DEFINITION 66. — *Given a normal program P, define an atomic normal program*

$$\mathrm{Tr}_{\mathrm{AT}}(P) = \mathrm{Tr}_{\mathrm{SUPP}}(P) \cup \mathrm{Tr}_{\mathrm{CTR}}(P) \cup \mathrm{Tr}_{\mathrm{MAX}}(P) \cup \mathrm{Tr}_{\mathrm{MIN}}(P)$$

and its visible part $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{AT}}(P)) = \mathrm{Hb}_{\mathrm{v}}(P)$.

By inspecting the four parts of $\mathrm{Tr}_{\mathrm{AT}}(P)$ once more, we note that $\mathrm{Tr}_{\mathrm{AT}}(P)$ can be formed in a very systematic fashion by generating certain rules for each $r \in P$ and $\mathsf{a} \in \mathrm{Hb}(P)$. However, $\mathrm{Tr}_{\mathrm{AT}}$ is not modular in the sense defined in Section 3.3. A source of non-modularity is hidden in the number of bits ∇P involved in $\mathrm{Tr}_{\mathrm{AT}}(P)$. Given two programs P and Q satisfying module conditions M1–M4, it is still possible that $\nabla P < \nabla(P \cup Q)$ and $\nabla Q < \nabla(P \cup Q)$. As a consequence, the counters involved in $\mathrm{Tr}_{\mathrm{AT}}(P)$ and $\mathrm{Tr}_{\mathrm{AT}}(Q)$ are based on too few bits, which implies that $\mathrm{Tr}_{\mathrm{AT}}(P)$ and $\mathrm{Tr}_{\mathrm{AT}}(Q)$ cannot be joined together in order to form the translation $\mathrm{Tr}_{\mathrm{AT}}(P \cup Q)$.

EXAMPLE 67. — Due to high number of rules generated by $\mathrm{Tr}_{\mathrm{AT}}$ we have to consider a logic program P consisting of only one rule $r = \mathsf{a} \leftarrow \mathsf{a}$.

The translation $\mathrm{Tr}_{\mathrm{SUPP}}(P)$ contains four rules: $\overline{\mathsf{a}} \leftarrow {\sim}\mathsf{a}$; $\mathsf{bt}(r) \leftarrow {\sim}\overline{\mathsf{a}}$; $\overline{\mathsf{bt}(r)} \leftarrow {\sim}\mathsf{bt}(r)$; and $\mathsf{a} \leftarrow {\sim}\overline{\mathsf{bt}(r)}$. Note that $\mathrm{Tr}_{\mathrm{SUPP}}(P)$ has two stable models $M_1 = \{\overline{\mathsf{a}}, \overline{\mathsf{bt}(r)}\}$ and $M_1 = \{\mathsf{a}, \mathsf{bt}(r)\}$. We need the rest of $\mathrm{Tr}_{\mathrm{AT}}(P)$ to exclude the latter. Since $\nabla P = 2$, the subprograms $\mathrm{SEL}_2(\mathsf{ctr}(\mathsf{a}), \overline{\mathsf{a}})$, $\mathrm{NXT}_2(\mathsf{ctr}(\mathsf{a}), \mathsf{nxt}(\mathsf{a}), \overline{\mathsf{a}})$, and $\mathrm{SEL}_2(\mathsf{ctr}(r), \overline{\mathsf{bt}(r)})$ of $\mathrm{Tr}_{\mathrm{CTR}}(P)$ aim to select 2-bit values for $\#\mathsf{a}$ and $\#\mathsf{r}$, but only when $\overline{\mathsf{a}}$ and $\overline{\mathsf{bt}(r)}$ are false. Otherwise, these counters remain inactive.

The program $\mathrm{Tr}_{\mathrm{MAX}}(P)$ consists of two subprograms $\mathrm{LT}_2(\mathsf{ctr}(r), \mathsf{nxt}(\mathsf{a}), \overline{\mathsf{bt}(r)})$ and $\mathrm{EQ}_2(\mathsf{ctr}(r), \mathsf{nxt}(\mathsf{a}), \overline{\mathsf{bt}(r)})$ plus the rules $\mathsf{x} \leftarrow {\sim}\mathsf{x}, {\sim}\overline{\mathsf{bt}(r)}, {\sim}\mathsf{lt}(\mathsf{ctr}(r), \mathsf{nxt}(\mathsf{a}))_1$; $\mathsf{max}(r) \leftarrow {\sim}\mathsf{bt}(r), {\sim}\mathsf{eq}(\mathsf{ctr}(r), \mathsf{nxt}(\mathsf{a}))$; and $\mathsf{x} \leftarrow {\sim}\mathsf{x}, {\sim}\overline{\mathsf{bt}(r)}, {\sim}\mathsf{max}(r)$. They compare $\#r$ and $\#\mathsf{a}+1$ (modulo $2^2 = 4$) and the net effect of the constraints is that $\#r = \#\mathsf{a} + 1$. Similarly, the translation $\mathrm{Tr}_{\mathrm{MIN}}(P)$ comprises of $\mathrm{LT}_2(\mathsf{ctr}(r), \mathsf{ctr}(\mathsf{a}), \overline{\mathsf{bt}(r)})$ and $\mathrm{EQ}_2(\mathsf{ctr}(r), \mathsf{ctr}(\mathsf{a}), \overline{\mathsf{bt}(r)})$ augmented by $\mathsf{y} \leftarrow {\sim}\mathsf{y}, {\sim}\overline{\mathsf{bt}(r)}, {\sim}\mathsf{lt}(\mathsf{ctr}(r), \mathsf{ctr}(\mathsf{a}))_1$; $\mathsf{min}(\mathrm{H}(r)) \leftarrow {\sim}\mathsf{bt}(r), {\sim}\mathsf{eq}(\mathsf{ctr}(r), \mathsf{ctr}(\mathsf{a}))$; and $\mathsf{y} \leftarrow {\sim}\mathsf{y}, {\sim}\overline{\mathsf{a}}, {\sim}\mathsf{min}(\mathsf{a})$ and they effectively state that $\#r = \#\mathsf{a}$. Note that in analogy to $\mathrm{Tr}_{\mathrm{CTR}}(P)$, the rules of $\mathrm{Tr}_{\mathrm{MAX}}(P)$ and $\mathrm{Tr}_{\mathrm{MIN}}(P)$ are activated only when $\overline{\mathsf{a}}$ and $\overline{\mathsf{bt}(r)}$ are false; or equivalently a and $\mathsf{bt}(r)$ are true by the structure of $\mathrm{Tr}_{\mathrm{SUPP}}(P)$. Consequently, M_2 becomes unstable as $\#r = \#\mathsf{a} + 1$ and $\#r = \#\mathsf{a}$ cannot be satisfied simultaneously. □

6.2.3. *Correctness of the translation function* $\mathrm{Tr}_{\mathrm{AT}}$

The correctness of $\mathrm{Tr}_{\mathrm{AT}}$ is addressed next. In order to describe the correspondence between stable models, the following definitions make explicit how a stable model M of a normal program P can be extended to a stable model N of the translation $\mathrm{Tr}_{\mathrm{AT}}(P)$. This is because $\mathrm{Tr}_{\mathrm{AT}}(P)$ involves many new atoms, the truth values of which have to be determined. First of all, we deal with atoms that are essentially defined by the rules of $\mathrm{Tr}_{\mathrm{SUPP}}(P)$ and define the respective extension operator $\mathrm{Ext}_{\mathrm{SUPP}}(P, M)$ for P and M below. Recall that in addition to reproducing

M, this part of the translation is responsible for defining the complementary atoms $\overline{\mathsf{a}}$, for which $\mathsf{a} \in \mathrm{Hb}(P)$, and the atoms $\mathsf{bt}(r)$ and $\overline{\mathsf{bt}(r)}$, which detect the satisfaction of $\mathrm{B}(r)$ for rules $r \in P$. Out of these atoms, the ones included in the set $\mathrm{Ext}_{\mathrm{SUPP}}(P, M)$ defined below are supposed be true in the corresponding stable model N of $\mathrm{Tr}_{\mathrm{AT}}(P)$.

DEFINITION 68. — *For a normal program P and an interpretation M of P, define an extension operator by setting* $\mathrm{Ext}_{\mathrm{SUPP}}(P, M) = M \cup \{\overline{\mathsf{a}} \mid \mathsf{a} \in \mathrm{Hb}(P) - M\} \cup \{\mathsf{bt}(r) \mid r \in \mathrm{SR}(P, M)\} \cup \{\overline{\mathsf{bt}(r)} \mid r \in P - \mathrm{SR}(P, M)\}$.

By the following definition, we introduce similar extension operators for the other parts of $\mathrm{Tr}_{\mathrm{AT}}(P)$. For instance, the rules in $\mathrm{Tr}_{\mathrm{CTR}}(P)$ are responsible for selecting correct values for the counters whose purpose is to capture the unique level numbering $\#$ w.r.t. M. As a result, the atoms in $\mathrm{Ext}_{\mathrm{CTR}}(P, M, \#)$ ought to be marked true in N. The last two parts of the translation contribute atoms involved in the constraints on the values of the counters, which implement the maximization/minimization principles from Definition 50. Again, the respective extension operators $\mathrm{Ext}_{\mathrm{MAX}}$ and $\mathrm{Ext}_{\mathrm{MIN}}$ determine which atoms evaluate to true given P, M, and $\#$.

DEFINITION 69. — *For a normal program P, an interpretation M of P, and a function $\# : M \cup \mathrm{SR}(P, M) \to \{0, \ldots, 2^{\nabla P} - 1\}$, define the following operators:*

$$\mathrm{Ext}_{\mathrm{CTR}}(P, M, \#) = \bigcup_{\mathsf{a} \in M} \mathrm{AT}^{\mathrm{ctr}}_{\nabla P}(\mathsf{ctr}(\mathsf{a}), \#\mathsf{a}) \cup \bigcup_{\mathsf{a} \in M} \mathrm{AT}^{\mathrm{ctr}}_{\nabla P}(\mathsf{nxt}(\mathsf{a}), \#\mathsf{a} + 1 \bmod 2^{\nabla P}) \cup \bigcup_{r \in \mathrm{SR}(P,M) \text{ and } \mathrm{B}^+(r) = \emptyset} \mathrm{AT}^{\mathrm{ctr}}_{\nabla P}(\mathsf{ctr}(r), 1) \cup \bigcup_{r \in \mathrm{SR}(P,M) \text{ and } \mathrm{B}^+(r) \neq \emptyset} \mathrm{AT}^{\mathrm{ctr}}_{\nabla P}(\mathsf{ctr}(r), \#r). \quad (23)$$

$$\mathrm{Ext}_{\mathrm{MAX}}(P, M, \#) = \{\mathsf{max}(r) \mid r \in \mathrm{SR}(P, M) \text{ and } \mathrm{B}^+(r) \neq \emptyset\} \cup \bigcup_{r \in \mathrm{SR}(P,M) \text{ and } \mathsf{a} \in \mathrm{B}^+(r)} \mathrm{AT}^{\mathrm{lt}}_{\nabla P}(\mathsf{ctr}(r), \#r, \mathsf{nxt}(\mathsf{a}), \#\mathsf{a} + 1 \bmod 2^{\nabla P}) \cup \bigcup_{r \in \mathrm{SR}(P,M) \text{ and } \mathsf{a} \in \mathrm{B}^+(r)} \mathrm{AT}^{\mathrm{eq}}_{\nabla P}(\mathsf{ctr}(r), \#r, \mathsf{nxt}(\mathsf{a}), \#\mathsf{a} + 1 \bmod 2^{\nabla P}). \quad (24)$$

$$\mathrm{Ext}_{\mathrm{MIN}}(P, M, \#) = \{\mathsf{min}(\mathsf{a}) \mid \mathsf{a} \in M\} \cup \bigcup_{r \in \mathrm{SR}(P,M)} \mathrm{AT}^{\mathrm{lt}}_{\nabla P}(\mathsf{ctr}(r), \#r, \mathsf{ctr}(\mathrm{H}(r)), \#\mathrm{H}(r)) \cup \bigcup_{r \in \mathrm{SR}(P,M)} \mathrm{AT}^{\mathrm{eq}}_{\nabla P}(\mathsf{ctr}(r), \#r, \mathsf{ctr}(\mathrm{H}(r)), \#\mathrm{H}(r)). \quad (25)$$

The four extensions operators introduced so far are combined into one extension operator for the whole translation $\mathrm{Tr}_{\mathrm{AT}}(P)$. It should be yet emphasized that the four sets of atoms involved in Definition 70 are disjoint.

DEFINITION 70. — *For a normal program P, an interpretation $M \subseteq \mathrm{Hb}(P)$ of P, and a function $\# : M \cup \mathrm{SR}(P, M) \to \{0, \ldots, 2^{\nabla P} - 1\}$, define* $\mathrm{Ext}_{\mathrm{AT}}(P, M, \#) = \mathrm{Ext}_{\mathrm{SUPP}}(P, M) \cup \mathrm{Ext}_{\mathrm{CTR}}(P, M, \#) \cup \mathrm{Ext}_{\mathrm{MAX}}(P, M, \#) \cup \mathrm{Ext}_{\mathrm{MIN}}(P, M, \#)$.

The correctness of the translation function $\mathrm{Tr}_{\mathrm{AT}}$ is addressed in Propositions 71 and 73 as well as Theorem 74.

PROPOSITION 71 ([JAN 03B]). — *Let P be a normal program. If M is a stable model of P and $\#$ is the corresponding level numbering w.r.t. M, then the interpretation $N = \mathrm{Ext}_{\mathrm{AT}}(P, M, \#)$ is a stable model of $\mathrm{Tr}_{\mathrm{AT}}(P)$ such that $M = N \cap \mathrm{Hb}(P)$.*

DEFINITION 72. — *Let P be a normal program, $N \subseteq \mathrm{Hb}(\mathrm{Tr}_{\mathrm{AT}}(P))$ an interpretation of the translation $\mathrm{Tr}_{\mathrm{AT}}(P)$, and $M = N \cap \mathrm{Hb}(P)$. Define a function $\# : M \cup \mathrm{SR}(P, M) \to \{0, \ldots, 2^{\nabla P} - 1\}$ by setting*

1) $\#\mathsf{a} = \mathrm{val}_{\nabla P}(\mathsf{ctr}(\mathsf{a}), N)$ for atoms $\mathsf{a} \in M$, and

2) $\#r = \mathrm{val}_{\nabla P}(\mathsf{ctr}(r), N)$ for rules $r \in \mathrm{SR}(P, M)$.

PROPOSITION 73 ([JAN 03B]). — *Let P be a normal program. If N is a stable model of the translation $\mathrm{Tr}_{\mathrm{AT}}(P)$, then $M = N \cap \mathrm{Hb}(P)$ is a stable model of P and $N = \mathrm{Ext}_{\mathrm{AT}}(P, M, \#)$ where $\#$ is defined as in Definition 72.*

THEOREM 74. — $\mathcal{P} \leq_{\mathrm{PF}} \mathcal{A}$.

However, due to the size and intricacy of $\mathrm{Tr}_{\mathrm{AT}}$, we skip the proof of correctness that can be found in [JAN 03b]. An important concept used in the proofs is the one of *local stability* given in Definition 75 below. As established in Theorem 76, atomic programs lend themselves to localizing the fixed point condition behind stable models. Consequently, the proofs for Propositions 71 and 73 can be established modularly.

DEFINITION 75. — *An interpretation I is **locally stable** w.r.t. a normal program P if and only if $I \cap \mathrm{H}(P) = \mathrm{LM}(P^I)$.*

THEOREM 76. — *Let $P_1, \ldots, P_n$ be **atomic** normal programs such that the sets of head atoms $\mathrm{H}(P_1), \ldots, \mathrm{H}(P_n)$ form a partition of $\mathrm{H}(P)$ for $P = \bigcup_{i=1}^{n} P_i$.*

For any $S \subseteq \{1, \ldots, n\}$, an interpretation $M \subseteq \mathrm{H}(P)$ is locally stable w.r.t. $P_S = \bigcup_{i \in S} P_i \iff M$ is locally stable w.r.t. P_i for every $i \in S$.

Moreover, an interpretation $M \subseteq \mathrm{Hb}(P)$ is a stable model of $P \iff M \subseteq \mathrm{H}(P)$ and M is locally stable w.r.t. P_i for every $i \in \{1, \ldots, n\}$.

In analogy to the classes of positive programs, the four classes of normal programs reside in the same expressiveness class if measured by the existence of polynomial and faithful translation function, i.e. the relation $\leq_{\mathrm{FM}}$.

COROLLARY 77. — $\mathcal{A} =_{\mathrm{PF}} \mathcal{U} =_{\mathrm{PF}} \mathcal{B} =_{\mathrm{PF}} \mathcal{P}$.

On the other hand, let us consider any class $\mathcal{C} \in \{\mathcal{A}, \mathcal{U}, \mathcal{B}, \mathcal{P}\}$. It follows by Proposition 19 that $\mathcal{C}^+ \leq_{\mathrm{PF}} \mathcal{C}$ and by Theorem 37 that $\mathcal{C} \not\leq_{\mathrm{PF}} \mathcal{C}^+$. Thus $\mathcal{C}^+ <_{\mathrm{PF}} \mathcal{C}$ holds for all representatives of the two expressiveness classes. The resulting hierarchy of classes of logic programs is illustrated in Figure 2. The relationships holding in this hierarchy remain in force even if we resort to weaker notions of faithfulness corresponding to equivalence relations addressed in Proposition 13.

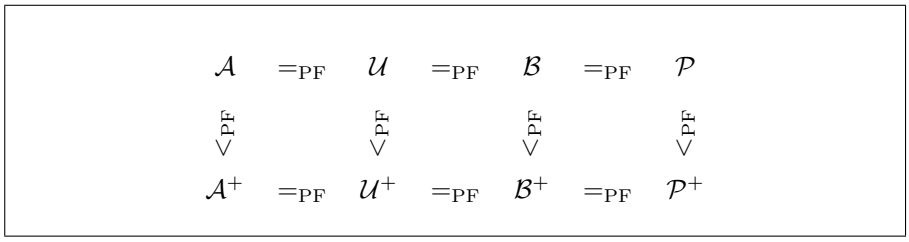

Figure 2. *Expressive power hierarchy based on polynomial and faithful (PF) translation functions*

6.3. *Propositional theories revisited*

In general, it is very challenging to translate a normal program P into a set of clauses so that a bijective correspondence of models is obtained. For instance, the approach by Ben-Eliyahu and Dechter [BEN 94] is based on a transformation that is clearly polynomial, but the produced set of clauses may possess multiple models corresponding to one stable model of P. However, atomic programs provide a promising intermediary representation that is relatively straightforward to translate into clauses. Here we can apply Clark's program completion as established by Fages [FAG 94], but new atoms have to be introduced by the translation function $\mathrm{Tr}_{\mathrm{CL}}$ in order to keep the translation function linear; or even polynomial in the first place.

DEFINITION 78. — *For an atomic normal program $P \in \mathcal{A}$ and an atom $\mathsf{a} \in \mathrm{Hb}(P)$, let $\mathrm{Def}_P(\mathsf{a}) = \{r \in P \mid \mathrm{H}(r) = \mathsf{a}\}$ and define the set of clauses*

$$\begin{aligned}\mathrm{Tr}_{\mathrm{CL}}(\mathsf{a}, P) = {} & \{\{\mathsf{a}, \neg\mathsf{bt}(r)\} \mid \mathsf{a} \in \mathrm{Hb}(P) \text{ and } r \in \mathrm{Def}_P(\mathsf{a})\} \cup \\ & \{\{\neg\mathsf{a}\} \cup \{\mathsf{bt}(r) \mid r \in \mathrm{Def}_P(\mathsf{a})\} \mid \mathsf{a} \in \mathrm{Hb}(P)\} \cup \\ & \{\{\mathsf{bt}(r)\} \cup \mathrm{B}^-(r) \mid r \in \mathrm{Def}_P(\mathsf{a})\} \cup \\ & \{\{\neg\mathsf{bt}(r), \neg\mathsf{c}\} \mid r \in \mathrm{Def}_P(\mathsf{a}) \text{ and } \mathsf{c} \in \mathrm{B}^-(r)\}\end{aligned}$$

where $\mathsf{bt}(r)$ is a new atom for each $r \in P$ and $\mathrm{Tr}_{\mathrm{CL}}(P) = \bigcup_{\mathsf{a} \in \mathrm{Hb}(P)} \mathrm{Tr}_{\mathrm{CL}}(\mathsf{a}, P)$ such that $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{CL}}(P)) = \mathrm{Hb}_{\mathrm{v}}(P)$, $\mathrm{Hb}_{\mathrm{h}}(\mathrm{Tr}_{\mathrm{CL}}(P)) = \mathrm{Hb}_{\mathrm{h}}(P) \cup \{\mathsf{bt}(r) \mid r \in P\}$.

We note that every atom of $\mathrm{Hb}(P)$ appears in $\mathrm{Tr}_{\mathrm{CL}}(P)$ so that $\mathrm{Hb}_{\mathrm{a}}(\mathrm{Tr}_{\mathrm{CL}}(P)) = \emptyset$. The intuitive reading of $\mathsf{bt}(r)$ is the same as in Section 6.2, i.e. $\mathsf{bt}(r)$ is supposed

to be true whenever the body of the rule r is true. Roughly speaking, the clauses in the translation ensure that every atom $\mathsf{a} \in \mathrm{Hb}(P)$ is logically equivalent to the disjunction of all bodies of rules $r \in P$ with $\mathrm{H}(r) = \mathsf{a}$. More precisely, clauses of the first two kinds in $\mathrm{Tr}_{\mathrm{CL}}(\mathsf{a}, P)$ enforce the equivalence of each $\mathsf{a} \in \mathrm{Hb}(P)$ with the disjunction $\bigvee\{\mathsf{bt}(r) \mid r \in \mathrm{Def}_P(\mathsf{a})\}$. On the other hand, each disjunct $\mathsf{bt}(r)$ is made equivalent to the conjunction of negative (classical) literals $\bigwedge\{\neg\mathsf{c} \mid \mathsf{c} \in \mathrm{B}^-(P)\}$ by clauses of the last two kinds in $\mathrm{Tr}_{\mathrm{CL}}(\mathsf{a}, P)$. The net effect is Clark's completion for each $\mathsf{a} \in \mathrm{Hb}(P)$. This leads to a tight correspondence of models as described next.

DEFINITION 79. — *Given an interpretation $I \subseteq \mathrm{Hb}(P)$ of $P \in \mathcal{A}$, define an extension operator* $\mathrm{Ext}_{\mathrm{CL}}(P, I) = I \cup \{\mathsf{bt}(r) \mid r \in \mathrm{SR}(P, I)\}$.

PROPOSITION 80 ([JAN 03B]). — *Let P be an atomic normal program. If $M \subseteq \mathrm{Hb}(P)$ is a supported model of P, then $N = \mathrm{Ext}_{\mathrm{CL}}(P, M)$ is a (classical) model of $\mathrm{Tr}_{\mathrm{CL}}(P)$ such that $M = N \cap \mathrm{Hb}(P)$.*

PROPOSITION 81 ([JAN 03B]). — *Let P be an atomic normal program. If an interpretation $N \subseteq \mathrm{Hb}(\mathrm{Tr}_{\mathrm{CL}}(P))$ is a (classical) model of $\mathrm{Tr}_{\mathrm{CL}}(P)$, then $M = N \cap \mathrm{Hb}(P)$ is a supported model of P such that $N = \mathrm{Ext}_{\mathrm{CL}}(P, M)$.*

EXAMPLE 82. — Consider a logic program P which consists of two rules $r_1 = \mathsf{a} \leftarrow {\sim}\mathsf{a}$ and $r_2 = \mathsf{a} \leftarrow {\sim}\mathsf{b}$ and has unique stable model $M = \{\mathsf{a}\}$. The translation $\mathrm{Tr}_{\mathrm{CL}}(P)$ contains clauses $\{\mathsf{a}, \neg\mathsf{bt}(r_1)\}$, $\{\mathsf{a}, \neg\mathsf{bt}(r_2)\}$, $\{\neg\mathsf{a}, \mathsf{bt}(r_1), \mathsf{bt}(r_2)\}$, $\{\neg\mathsf{b}\}$, $\{\mathsf{bt}(r_1), \mathsf{a}\}$, $\{\neg\mathsf{bt}(r_1), \neg\mathsf{a}\}$, $\{\mathsf{bt}(r_2), \mathsf{b}\}$, and $\{\neg\mathsf{bt}(r_2), \neg\mathsf{b}\}$. There is a unique classical model $N = \{\mathsf{a}, \mathsf{bt}(r_2)\}$ of $\mathrm{Tr}_{\mathrm{CL}}(P)$, as interpretations are restricted to the Herbrand base $\mathrm{Hb}(\mathrm{Tr}_{\mathrm{CL}}(P)) = \{\mathsf{a}, \mathsf{b}, \mathsf{bt}(r_1), \mathsf{bt}(r_2)\}$. □

The translation function $\mathrm{Tr}_{\mathrm{CL}}$ is clearly non-modular, since the clauses of the type $\{\neg\mathsf{a}\} \cup \{\mathsf{bt}(r) \mid r \in \mathrm{Def}_P(\mathsf{a})\}$ create a dependency between rules possessing the same head a. Let us then address polynomiality and faithfulness as suggested by the one-to-one correspondence obtained in Example 82.

PROPOSITION 83. — *Let P be an atomic normal program. Then $M \subseteq \mathrm{Hb}(P)$ is a stable model of P if and only if M is a supported model of P.*

PROOF. — ($\Longrightarrow$) This is shown by Marek and Subrahmanian [MAR 92] for normal programs (including atomic ones). ($\Longleftarrow$) Let M be a supported model of P. Let us define a function $\#$ from $M \cup \mathrm{SR}(P, M)$ to $\mathbb{N}$ such that $\#\mathsf{a} = 1$ for all $\mathsf{a} \in M$ and $\#r = 1$ for all $r \in \mathrm{SR}(P, M)$. Since P is atomic, we have $\mathrm{B}^+(r) = \emptyset$ for every $r \in P$ and it is easy to inspect from Definition 50 that $\#$ is a level numbering w.r.t. M. Thus M is a stable model of P by Theorem 55. ■

THEOREM 84. — $\mathcal{A} \leq_{\mathrm{PF}} \mathcal{PT}$.

PROOF. — An atomic rule $r = \mathsf{a} \leftarrow {\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m$ consists of $3m + 2$ symbols if each atom counts as one symbol and one symbol is reserved for separating it from other rules. The translation function $\mathrm{Tr}_{\mathrm{CL}}$ translates r effectively into clauses $\{\mathsf{a}, \neg\mathsf{bt}(r)\}$, $\{\mathsf{bt}(r), \mathsf{c}_1, \ldots, \mathsf{c}_m\}$, $\{\neg\mathsf{bt}(r), \neg\mathsf{c}_1\}$, …, and $\{\neg\mathsf{bt}(r), \neg\mathsf{c}_m\}$ which contain $10m + 11$ symbols (including separating commas). In addition, the rule r contributes one literal to $\{\neg\mathsf{a}\} \cup \{\mathsf{bt}(r') \mid r' \in \mathrm{Def}_P(\mathsf{a})\}$ which produces two additional

symbols for r and 4 symbols for each $\mathsf{a} \in \mathrm{Hb}(P)$. The translation $\mathrm{Tr}_{\mathrm{CL}}(P)$ can be produced by going through the rules of P, creating the clauses and keeping an account of atoms that appear as heads in the rules. The clause $\{\neg\mathsf{a}\} \cup \{\mathsf{bt}(r') \mid r' \in \mathrm{Def}_P(\mathsf{a})\}$ needs to be created for such atoms. Thus we conclude $\mathrm{Tr}_{\mathrm{CL}}$ to be linear/polynomial.

To establish the faithfulness of $\mathrm{Tr}_{\mathrm{CL}}$, let P be an atomic normal program. Note that $\mathrm{Hb}(P) \subseteq \mathrm{Hb}(\mathrm{Tr}_{\mathrm{CL}}(P))$ and $\mathrm{Hb}_{\mathrm{v}}(\mathrm{Tr}_{\mathrm{CL}}(P)) = \mathrm{Hb}_{\mathrm{v}}(P)$ hold directly by Definition 78. It follows by Propositions 83 and 80 that there is an extension function $\mathrm{Ext}_{\mathrm{CL}} : \mathrm{SM}(P) \to \mathrm{CM}(\mathrm{Tr}_{\mathrm{CL}}(P))$ that maps $M \in \mathrm{SM}(P)$ into $N = \mathrm{Ext}_{\mathrm{CL}}(P, M)$ included in $\mathrm{CM}(\mathrm{Tr}_{\mathrm{CL}}(P))$ such that $M = N \cap \mathrm{Hb}(P)$. Moreover, Propositions 81 and 83 imply that that if $N \in \mathrm{CM}(\mathrm{Tr}_{\mathrm{CL}}(P))$, then $M = N \cap \mathrm{Hb}(P) \in \mathrm{SM}(P)$ and $N = \mathrm{Ext}_{\mathrm{CL}}(P, M)$. Thus we may conclude $\mathrm{Tr}_{\mathrm{CL}}$ to be faithful by Proposition 20. ■

COROLLARY 85. — $\mathcal{PT} =_{\mathrm{PF}} \mathcal{A} =_{\mathrm{PF}} \mathcal{U} =_{\mathrm{PF}} \mathcal{B} =_{\mathrm{PF}} \mathcal{P}$.

COROLLARY 86. — $\mathcal{C}^+ <_{\mathrm{PF}} \mathcal{PT}$ *holds for any* $\mathcal{C}^+ \in \{\mathcal{A}^+, \mathcal{U}^+, \mathcal{B}^+, \mathcal{P}^+\}$.

7. Related work

The classification method presented in Section 3.4 accommodates the one proposed for non-monotonic logics [JAN 99b, JAN 03a, JAN 00a] to the case of logic programs. These classification methods are analogous, but there are also major differences. The semantics of a *non-monotonic theory* is determined by a set of *extensions/expansions* which are typically propositionally closed theories[11] rather than a set of interpretations/models. This makes the notions of faithfulness somewhat incompatible although bijective correspondences are aimed by both of them. The concepts of modularity are also different because the primary objects of study, i.e. non-monotonic theories and logic programs, are epitomized differently.

We should also comment on the major changes made to earlier versions of the method presented in [JAN 00b, JAN 01] in which the systematic analysis of logic programs was initiated. Firstly, the notion of modularity presented in Section 3.3 is more fine-grained due to module conditions M1–M4 given in Definition 16. More precisely, the condition (7) is supposed to hold in limited context while P and Q are assumed arbitrary in [JAN 00b]. A further difference in modularity is that $\mathrm{Tr}(P) = P$ for all $P \in \mathcal{C}$ when $\mathrm{Tr} : \mathcal{C}' \to \mathcal{C}$ and $\mathcal{C} \subset \mathcal{C}'$, i.e. $\mathcal{C}$ is a syntactic subclass of $\mathcal{C}'$. Although this leads to analogous intranslatability results [JAN 00b] it is impossible to make comparisons with syntactically different classes such as $\mathcal{PT}$ distinguished in Section 3.1. This is why we resort to a weaker notion of modularity. A further difference concerns the notion of faithfulness proposed in Section 3.3. It is weaker than the one used in [JAN 00b] because the visibility of atoms is taken fully into account. These kinds of weakenings are in favor of intranslatability results which are strengthened. However, the resulting expressive power hierarchy is not affected by the tunings made. But, as discussed in Section 4.2, certain strict relationships in the hierarchy may cease to hold if still weaker notions of faithfulness are introduced.

11. Recall that a propositionally closed theory is fully determined by the set of its models.

Module systems are used in programming languages to manage the complexity of programs as well as to enforce good programming practice. Logic programming is not an exception in this respect as several proposals for modular composition of logic programs have been made; see [BUG 94] for a comprehensive study. For instance, Maher [MAH 93] distinguishes *internal* predicates from *imported* and *exported* predicates and imposes conditions on program composition under *perfect* model semantics. In our terminology and in the propositional case, the first category forms the hidden part of the Herbrand base while the latter two categories form its visible part. For us, the distinction of imported and exported atoms is not important and the module conditions M1–M4 of Definition 16 are much weaker. Again, this favors our negative results as stated above. Bugliesi et al. [BUG 94] present a very similar conceptualization of program modules due to Gaifman and Shapiro accompanied by an algebraic theory of program composition. Etalle and Gabbrielli [ETA 96] define modules in analogy to Definition 6 (assuming the propositional case) but leave the hidden part of the Herbrand base implicit. Their only module condition is very close to ours: the joint atoms in $\mathrm{Hb}(P) \cap \mathrm{Hb}(Q)$ must be contained in $\mathrm{Hb}_{\mathrm{v}}(P) \cap \mathrm{Hb}_{\mathrm{v}}(Q)$. One obvious difference is that P and Q are not required to be disjoint.

Antoniou et al. [ANT 01] apply a modularity condition when developing normal forms for Nute's *defeasible logic* [NUT 94]. Although defeasible logic is based on a completely different semantics, its rule-based syntax makes it reminiscent of normal programs. Antoniou et al. consider a translation function Tr to be *correct*, if $D \equiv_{L(D)} \mathrm{Tr}(D)$ for every D.[12] This is somewhat analogous to our approach, but there is no account of visibility of atoms and the semantics of defeasible logic assigns a unique set of conclusions to each theory. A further property addressed in [ANT 01] is *incrementality* defined by $\mathrm{Tr}(D_1 \cup D_2) \equiv_{L(D_1) \cup L(D_2)} \mathrm{Tr}(D_1) \cup \mathrm{Tr}(D_2)$. This is already very close to (7) in our definition of modularity. However, our notion is based on syntactical equality $=$ rather than semantical equivalence $\equiv_{\mathrm{v}}$. Moreover, there is no counterpart to module conditions in the approach by Antoniou et al. Actually, they reserve the term *modularity* for a stronger property described by $D_1 \cup D_2 \equiv_{L(D_1) \cup L(D_2)} D_1 \cup \mathrm{Tr}(D_2)$. In contrast to (7), such a condition is not applicable to translations between classes which are syntactically different. For example, consider the case that D_1 is a defeasible theory and $\mathrm{Tr}(D_2)$ is a propositional theory. The union $D_1 \cup \mathrm{Tr}(D_2)$ does not make sense.

As shown in Section 4.3, normal programs cannot be translated into sets of clauses in a faithful and modular way. Niemelä [NIE 99, Proposition 4.3] provides a formal counter-example in this respect, too, but the result is intentionally strengthened using much weaker notions of faithfulness and modularity. In spite of these intranslatability results, the composition of the translation functions $\mathrm{Tr}_{\mathrm{AT}}$ and $\mathrm{Tr}_{\mathrm{CL}}$ from Section 6 is sufficient to reduce normal logic programs into propositional theories. The resulting translation function is definitely not modular, but still highly structural so that actual translations can be computed in a systematic fashion. A transformation that would

12. Here $\equiv_{L(D)}$ denotes semantical equivalence, i.e., the theories yield exactly the same conclusions in the language $L(D)$ of D.

not introduce new atoms seems very unlikely in light of a recent complexity-theoretic argument by Lifschitz and Razborov [LIF 06]. On the other hand, Niemelä [NIE 99] presents the basic technique to encode propositional satisfiability problems in terms of normal logic programs. The translation function $\mathrm{Tr}_{\mathrm{LP}}$ presented is Section 4.3 is designed with stronger criteria (i.e. faithfulness and modularity) in mind.

Marek and Truszczyński [MAR 91] show that checking whether a normal logic program has a stable model forms an NP-complete decision problem which is analogous to the propositional satisfiability SAT problem [COO 71]. The translation functions $\mathrm{Tr}_{\mathrm{AT}}$ and $\mathrm{Tr}_{\mathrm{LP}}$ imply that the computational complexity of the former problem remains NP-complete under the three syntactic restrictions introduced in Section 2.1. This indicates that the expressive powers of the classes $\mathcal{A}$, $\mathcal{U}$, and $\mathcal{B}$ cannot be differentiated in terms of traditional complexity measures. This is mainly because the reducibilities involved in complexity results preserve only the yes/no answers to decision problems. In contrast, the relation $<_{\mathrm{PFM}}$ based on the existence of a polynomial, faithful and modular translation function enables us to detect strict differences.

Partial evaluation techniques have been introduced to rewrite rules of programs in a semantics preserving way. A good example in this respect is an approach by Brass and Dix [BRA 97]. They propose *equivalence transformations* for normal and *disjunctive* logic programs under the stable model semantics [GEL 91]. Partial evaluation is one of such transformations and particularly interesting from our point of view as it may decrease favorably the number of positive body literals. This takes place, e.g., when a rule $\mathsf{a} \leftarrow \mathsf{b}$ is replaced by $\mathsf{a} \leftarrow \sim\mathsf{c}$ and $\mathsf{a} \leftarrow \sim\mathsf{d}$ given that the definition of b consists of $\mathsf{b} \leftarrow \sim\mathsf{c}$ and $\mathsf{b} \leftarrow \sim\mathsf{d}$. A drawback is that partial evaluation may cause an exponential growth to the length of the program in the worst case. Nevertheless, partial evaluation preserves the Herbrand base of the program and eventually, it will produce an atomic normal program if applied as far as possible. E.g., Costantini et al. [COS 02] use partial evaluation for this purpose. In fact, partial evaluation (also known as *unfolding*) is admitted by a variety of semantics proposed for normal and disjunctive programs; see [ARA 95, BRA 97] for a collection of results in this respect. The more conventional case of definite programs with variables and function symbols is covered in the survey by Pettorossi and Proietti [PET 94].

Ben-Eliyahu and Dechter [BEN 94] study the possibilities of reducing *head-cycle-free* disjunctive logic programs, under the stable model semantics [GEL 91], to propositional theories. We restrict our attention to normal programs which form a special case of head-cycle-free disjunctive programs. Ben-Eliyahu and Dechter [BEN 94, Theorem 2.8] devise a characterization of stable models that resembles the one developed in Section 5. However, they impose weaker conditions on level numberings so that in contrast to Theorem 55, uniqueness cannot be guaranteed. The translation function $\mathrm{Tr}_{\mathrm{BD}}$ (called *translate-2* in [BEN 94]) produces a propositional theory $\mathrm{Tr}_{\mathrm{BD}}(P)$ that captures stable models in terms of classical ones. In particular, the fact that $\#\mathsf{a}$ equals to i for some atom $\mathsf{a} \in \mathrm{Hb}(P)$ is expressed by making a new atom $\mathsf{in}(\mathsf{a})_i$ true on the classical side. In contrast to the composition $\mathrm{Tr}_{\mathrm{AT}} \circ \mathrm{Tr}_{\mathrm{CL}}$, the translation function $\mathrm{Tr}_{\mathrm{BD}}$ does not necessarily yield a bijective correspondence between models as the

level numberings used by used by Ben-Eliyahu and Dechter lack uniqueness. Moreover, the language of P is not preserved by $\mathrm{Tr}_{\mathrm{BD}}$ as $\mathrm{Hb}(P) \cap \mathrm{Hb}(\mathrm{Tr}_{\mathrm{BD}}(P)) = \emptyset$. A further difference is that $||\mathrm{Tr}_{\mathrm{BD}}(P)||$ is quadratic in $||P||$ in the worst case. Our translation functions are more economical in this respect: $||\mathrm{Tr}_{\mathrm{CL}}(\mathrm{Tr}_{\mathrm{AT}}(P))||$ is of the order of $||P|| \times \nabla P$, as the binary encoding of level numbers is used. This aspect of $\mathrm{Tr}_{\mathrm{AT}} \circ \mathrm{Tr}_{\mathrm{CL}}$ makes a difference with respect to a recent translation proposed by Lin and Zhao [LIN 03]. Their translation function is faithful but quadratic in $||P||$.

There are also other characterizations of stable models that are closely related to the one established in Section 5. Fages [FAG 94] calls an interpretation $I \subseteq \mathrm{Hb}(P)$ of a normal program P *well-supported* if and only if there exists a strict well-founded partial order $\prec$ on I such that for any atom $\mathsf{a} \in I$, there exists $r \in \mathrm{SR}(P, I)$ satisfying $\mathrm{H}(r) = \mathsf{a}$ and $\mathsf{b} \prec \mathsf{a}$ for all $\mathsf{b} \in \mathrm{B}^+(r)$. It follows that well-supported models of a normal program P are stable models of P, and vice versa [FAG 94]. Given a supported model M of P and the respective unique level numbering conforming to Definition 50, one can extract a strict well-founded partial order $\prec$ as follows: define $\mathsf{a} \prec \mathsf{b}$ if and only if $\mathsf{a} \in M$, $\mathsf{b} \in M$, and $\#\mathsf{a} < \#\mathsf{b}$. Furthermore, Fages distinguishes *positive order consistent* normal programs whose models are necessarily well-supported. As a consequence, the classical models of the completion of a program P [CLA 78], or the supported models of P, coincide with the stable models of P.

Babovich et al. [BAB 00] and Erdem and Lifschitz [ERD 03] generalize Fages' results by introducing the notion of *tightness* for logic programs. The tightness of a logic program P is defined relative to a set atoms $A \subseteq \mathrm{Hb}(P)$, which makes Fages' theorem applicable to a wider range of programs. To point out our contribution in this respect, we note that atomic normal programs are automatically positive order consistent, or *absolutely* tight [ERD 03]. Therefore, arbitrary normal programs can be transformed into absolutely tight ones by applying $\mathrm{Tr}_{\mathrm{AT}}$ from Section 6. Yet another approach [LIN 04] to deal with non-tight programs is to add *loop formulas* to the completion in order to exclude supported models which are not stable. This can be done gradually while computing classical models for the completion. As a drawback, the number of loop formulas can be exponential in the worst case. In contrast, $\mathrm{Tr}_{\mathrm{AT}} \circ \mathrm{Tr}_{\mathrm{CL}}$ provides a one-shot translation with a fixed length of the order of $||P|| \times \nabla P$.

As already discussed in Section 3.2, a basic notion of equivalence is obtained for a given class of programs $\mathcal{C}$ by requiring that programs possess the same stable models, i.e. $P \equiv Q$ iff $\mathrm{Sem}_{\mathcal{C}}(P) = \mathrm{Sem}_{\mathcal{C}}(Q)$. Lifschitz et al. [LIF 01] and later Turner [TUR 03] study a stronger condition, which involves an arbitrary context $R \in \mathcal{C}$ in which P and Q could be placed as subprograms. That is, P and Q are *strongly equivalent*, denoted by $P \equiv_{\mathrm{s}} Q$ iff for all $R \in \mathcal{C}$, $\mathrm{Sem}_{\mathcal{C}}(P \cup R) = \mathrm{Sem}_{\mathcal{C}}(Q \cup R)$. The equivalence relation $\equiv_{\mathrm{v}}$ introduced in Section 3.2 is more practically oriented, as it takes the visibility of atoms properly into account. It is justifiable to exclude atoms formalizing auxiliary concepts as concerns the equivalence of logic programs. This aspect is also present in Maher's early work [MAH 88] where a number of semantical concepts is projected with respect to the common Herbrand base of the programs being compared. But stable models [GEL 88] are not covered due to time of publication.

Quite recently, Woltran [WOL 04] has characterized relativized versions of $\equiv$ and $\equiv_s$ as regards visibility. There are also interesting connections between modularity and strong equivalence as pointed out by Ferraris and Cabalar [FER 05, CAB 05]. According to their results, modular translations from logic programs into propositional theories are enabled if their semantics is changed to be determined by *equilibrium models* that characterize strong equivalence.

```
node(1..n).
in(V1,V2) :- not out(V1,V2), node(V1;V2), V1!=V2.
out(V1,V2) :- not in(V1,V2), node(V1;V2), V1!=V2.
reach(V,V) :- node(V).
reach(V1,V3) :- in(V1,V2), reach(V2,V3),
                node(V1;V2;V3), V1!=V2, V1!=V3.
:- not reach(V1,V2), node(V1;V2).
```

Figure 3. *Normal logic program used in the reachability benchmark*

8. Experiments

In this section, we present prototype implementations of the translation functions $\mathrm{Tr}_{\mathrm{AT}}$ and $\mathrm{Tr}_{\mathrm{CL}}$ crafted in Sections 6.2 and 6.3, respectively. By combining these tools with SAT solvers we obtain a machinery which is sufficient to compute stable models for normal logic programs in practice. The main objective of this section is to make a preliminary comparison of this approach with existing answer set solvers using three benchmark problems. Our first benchmark deals with the reachability of nodes in graphs whereas the second is about checking the equivalence of normal logic programs. The computation of Hamiltonian circuits for randomly generated planar graphs is of interest in our last benchmark.

The implementation of our translation-based approach consists of two translators[13] called LP2ATOMIC and LP2SAT, which correspond to the two phases of the translation. The task of LP2ATOMIC is to translate away positive body atoms from a normal program given as input in the internal file format of SMODELS [SIM 02]; typically produced by the front-end LPARSE. The implementation includes certain optimizations not addressed in Section 6.2. For instance, *strongly connected components* (SCCs) are utilized to reduce the numbers of bits involved in binary counters and to avoid counters associated with rules in many cases. The latter translator, LP2SAT takes the output of LP2ATOMIC as its input and produces the completion for the program in the *DIMACS format* understood by most SAT solvers.

13. Please visit `http://www.tcs.hut.fi/Software/lp2sat/` for binaries and benchmarks.

All experiments reported in this section are run under the Debian GNU/Linux 2.4.26 operating system on a AMD Athlon XP 2000+ 1.67 GHz CPU with 1 GB memory. We use a variety of systems in the experiments: SMODELS [SIM 02], CMODELS [LIE 04], and ASSAT [LIN 04]; as well as combinations of LP2ATOMIC and LP2SAT with other solvers including three SAT solvers RELSAT [BAY 97], CHAFF[14] [MOS 01], and SIEGE[15]. The system LP2ATOMIC+SMODELS combines the two subsystems just to get an idea how much overhead results from the removal of positive body atoms. The three combinations LP2SAT+RELSAT, LP2SAT+CHAFF, and LP2SAT+SIEGE use both of our translators as well as the mentioned SAT solver as a back-end for the actual computation of stable models. Finally, we incorporate a strengthened well-founded reduction to these systems (the prefix WF+ is inserted to the name) by calling SMODELS to simplify the intermediate program representations before and after invoking LP2ATOMIC. The combined systems are implemented as shell scripts which are distributed in the same directory as binaries. We use three benchmarks to compare the performance of the systems as reported below.

Table 3. *Timings in seconds when computing all stable models*

Vertices	2	3	4	5
SMODELS	0.003	0.003	0.033	12
CMODELS	0.030	0.124	293	-
LP2ATOMIC+SMODELS	0.008	0.013	0.393	353
LP2SAT+CHAFF	0.009	0.023	1.670	-
LP2SAT+RELSAT	0.005	0.018	0.657	1879
WF+LP2SAT+RELSAT	0.013	0.018	0.562	1598
Models	1	18	1606	565080
SCCs with $\|\mathrm{H}(C)\| > 1$	0	3	4	5
Rules (LPARSE)	14	39	84	155
Rules (LP2ATOMIC)	18	240	664	1920
Clauses (LP2SAT)	36	818	2386	7642
Clauses (WF+LP2SAT)	10	553	1677	5971

– *Reachability problem* [JAN 04]. This benchmark is based on the complete directed graph D_n with n nodes and $n^2 - n$ arcs (there are no reflexive arcs). The problem is to find all subgraphs of D_n in which all vertices are still reachable from each other through the remaining arcs. The corresponding test program is given in the input syntax of LPARSE (see Figure 3). In the benchmark, the task is to compute all stable models of the program instantiated by LPARSE when n varies from 2 to 5. As a result, the number of SCCs involving *positive loops* increases. And in particular, the number stable models to be computed by the systems increases very rapidly. The results have been collected in Table 3. Our benchmark is really easy for SMODELS.

14. A very succesful SAT solver in competitions; see `http://www.satcompetition.org/`.
15. See `http://www.cs.sfu.ca/~loryan/personal/` for background.

However, the main objective here is to compare CMODELS with our approach as it is based on similar methodology. We did not use ASSAT nor SIEGE as they have been designed to compute only one model. Our results indicate that the systems based on LP2ATOMIC and LP2SAT appear to be faster than CMODELS. When $n = 5$, CMODELS exceeds the time limit of 24 hours and CHAFF runs out of memory (1 GB) as the back-end of LP2SAT. The systems perform differently when we compute only one stable model for the program and $n = 8$. The respective timings are 0.012, 0.043, $>10^4$, 0.80, 2.6 and 2.8 seconds for the systems in Table 3; and 0.020 seconds for ASSAT.

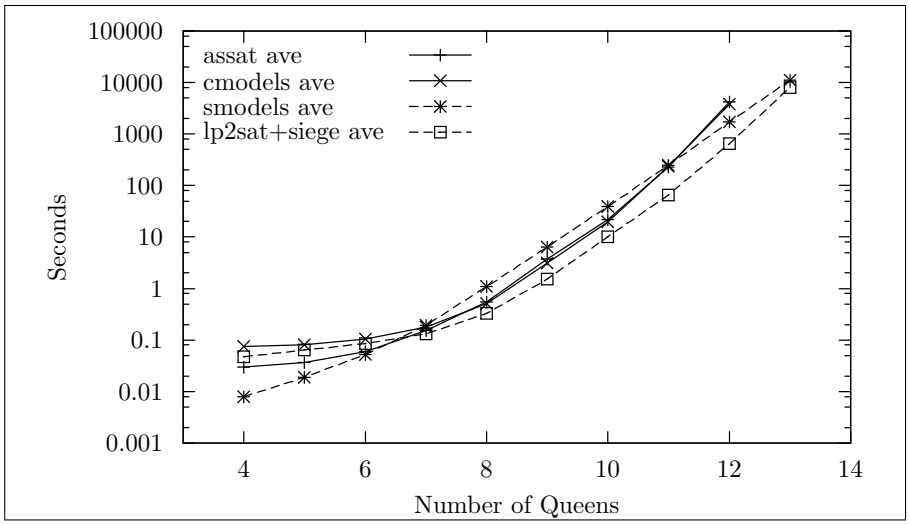

Figure 4. *Timings in seconds when showing the non-existence of stable models (averaged over 10 runs on the same instance)*

– *Verifying the equivalence of programs.* Our second benchmark is related to the famous n-queens problem where the task is to place n queens on an $n \times n$ chess board so that they do not threaten each other. The idea is to use two orthogonal variants of a program formulated by Niemelä [NIE 99]: one is based on a row-wise placement of the queens while the other uses columns. The task is to prove the equivalence of the two programs, which is achieved by (i) combining the programs using a specialized tool called LPEQ [JAN 02] and (ii) showing that the result does not have stable models. In the benchmark, the number of queens n is varied from 4 to 12. The running times of four systems have been plotted in Figure 4. ASSAT and CMODELS scale very similarly and run out of memory when $n = 13$. The SMODELS system is slightly better and LP2SAT+SIEGE turns out to be the best combination. Other SAT solvers lead to slower performance and the well-founded reduction does not really prune the program instances. SMODELS and LP2SAT+SIEGE can handle $n = 13$.

– *Hamiltonian circuits.* Our last benchmark concerns the problem of finding Hamiltonian circuits on random planar graphs generated generated by the Stanford Graph Base (SGB). Again, we have to slightly modify a program written by Niemelä

[NIE 99], since *choice rules* are not yet supported by LP2ATOMIC and LP2SAT. In the benchmark, the number of nodes is varied from 10 to 40. The results are illustrated in Figure 5. For small numbers of nodes, SMODELS outperforms the others, namely CMODELS, LP2SAT+SIEGE, and ASSAT in the order of improving performance. However, the average performance of SMODELS becomes worse as the number of nodes approaches 40 while the other three systems scale in a more robust way. Eventually, the performances of ASSAT and LP2SAT+SIEGE are almost identical. It is worth mentioning that the latter cannot be improved by using CHAFF as the back-end nor the well-founded reduction to simplify its input.

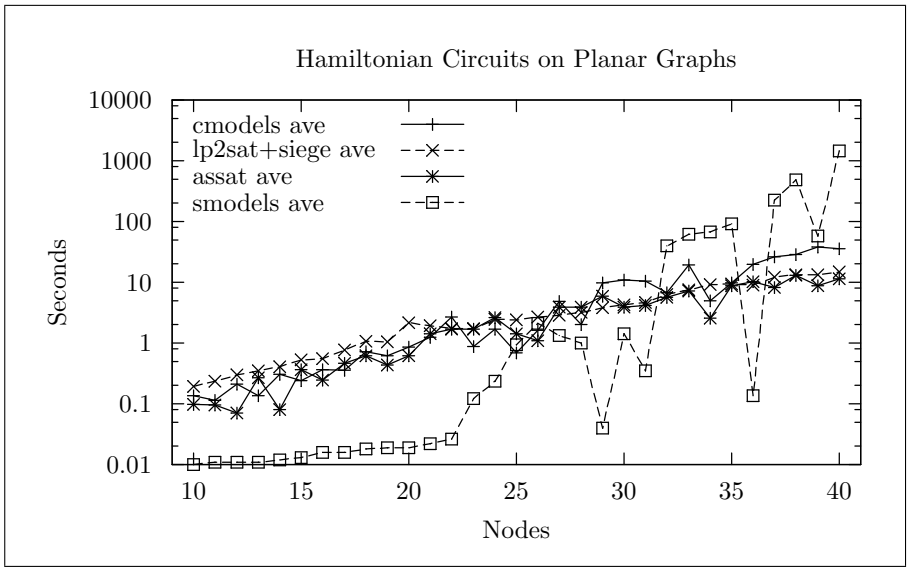

Figure 5. *Timings in seconds when computing one stable model (averaged over 100 runs on different instances)*

Our three benchmark problems involve basic reasoning tasks in answer set programming: (i) finding one stable model, (ii) showing the non-existence of stable models, and (iii) finding all stable models. The experiments reported above provide us the first indication that our translation-based approach is becoming competitive with other systems [LIE 04, LIN 04] employing SAT solvers for the computation of stable models. Our experience is that this holds mostly for the last two reasoning tasks given that the number of model candidates that have to be excluded is sufficient. Consequently, the other systems have to introduce loop formulas which eventually compensate the larger input program generated by LP2ATOMIC and LP2SAT in the beginning. However, if the task is to compute just one stable model for a program that has plenty of stable models, then ASSAT and CMODELS can be much faster as they benefit from the smaller initial representation of the program.

It is also very encouraging that in certain cases, the translation-based approach even outperforms a native ASP solver SMODELS [SIM 02] although SMODELS is of

its own league in the first experiment. In this case, the combination of LP2ATOMIC and SMODELS gives us more insight into the feasibility of removing positive positive body atoms from rules. When $n = 5$ the translation produced by LP2ATOMIC is roughly 15 times longer (in symbols) than the original program whereas a 30-fold increase in the running time of SMODELS is perceived. Thus the overall consequences of removing all positive body atoms seem quite negative which suggests us to study variants of $\mathrm{Tr}_{\mathrm{AT}}$ that preserve positive body atoms whenever possible.

9. Conclusions

This research started from the problem of reducing the number of positive subgoals in the bodies of rules of normal logic programs. To analyze this problem, we propose a classification method based on PFM translation functions in Section 3. The method is designed for the comparison of classes of logic programs on the basis of their expressive power and it is applied in Section 4 to the analysis of classes obtained by restricting the number of positive subgoals in rules. Retrospectively speaking, the method was obtained by adjusting the one presented in [JAN 00b, JAN 01] in many respects. In the new design, many objectives are settled: (i) the comparison of classes which may differ by either syntax or semantics is possible, (ii) the properties characterizing PFM translation functions are preserved under composition, and (iii) our preliminary expressiveness results [JAN 00b] remain valid. Moreover, the development of the underlying theory forced us to tackle with many important technical details such as visibility of atoms, mechanisms to extend Herbrand bases, notions of equivalence, and module conditions. Many of these ideas arose from our practical experience with answer set programming and experiments with existing implementations.

The expressiveness analysis reveals the main constituents of rule-based reasoning. In the simplest case, we have just sets of atomic rules $\mathsf{a} \leftarrow$ describing the state of the world; and no further inferences are possible. Unary rules enrich this setting by allowing *chained inferences* with rules, e.g. we can infer a using rules $\mathsf{a} \leftarrow \mathsf{b}$; $\mathsf{b} \leftarrow \mathsf{c}$; and $\mathsf{c} \leftarrow$. Binary rules incorporate conjunctive conditions to rule-based reasoning. For instance, one can derive a using $\mathsf{a} \leftarrow \mathsf{b}, \mathsf{c}$; $\mathsf{b} \leftarrow \mathsf{d}$; $\mathsf{c} \leftarrow \mathsf{d}$; and $\mathsf{d} \leftarrow$. Non-binary rules with more than two positive subgoals are easily reducible to these primitive forms, but our formal counter-examples indicate that binary and unary rules are not expressible in a faithful and modular way using unary and atomic rules, respectively, whatever number of rules is used. Moreover, the use of negation as failure in the bodies of rules does not affect this setting. Looking back to EPH illustrated in Figure 1, the number of positive body literals appears to be an essential syntactic restriction, as strict differences in expressive power can be established. It is also interesting to realize that propositional theories do not capture reasoning with (atomic) normal programs very easily, as shown in Section 4.3. There is also practical evidence for this, as many problems tend to be easier to formalize using rules rather than clauses.

The new characterization of stable models developed in Section 5 is based on uniquely determined level numberings. This contrasts with earlier characterizations

which enable the assignment of level numbers even in infinitely many ways. Due to the strong notion of faithfulness employed herein, unique level numberings are crucial for the main objective of Section 6, i.e. a polynomial and faithful translation of normal programs into atomic programs and propositional clauses. The translation function $\mathrm{Tr}_{\mathrm{AT}} \circ \mathrm{Tr}_{\mathrm{CL}}$ possesses a distinctive combination of features: (i) all finite normal programs P are covered, (ii) a bijective relationship of models is obtained, (iii) the Herbrand base $\mathrm{Hb}(P)$ is preserved, (iv) the length $||\mathrm{Tr}_{\mathrm{CL}}(\mathrm{Tr}_{\mathrm{AT}}(P))||$ as well as the translation time are of the order of $||P|| \times \log_2 |\mathrm{Hb}(P)|$, i.e. sub-quadratic, and (v) there is no need for incremental updating. We consider this as a breakthrough, since the best known transformations to date [BEN 94, LIN 03] are quadratic. A further implication of $\mathrm{Tr}_{\mathrm{AT}} \circ \mathrm{Tr}_{\mathrm{CL}}$ is that various closures of relations, such as the *transitive closure* (c.f. [ERD 01]), can be properly captured with classical models. This aspect is crucial in many applications involving e.g. some form of reachability.

The experiments reported in Section 8 are limited and by no means conclusive. For us, they serve as a proof of concept while demonstrating that our translation-based approach can be competitive in certain cases. Nevertheless, we are still pursuing for new optimizations techniques to be integrated to our translators. A further objective is to support the full SMODELS language involving cardinality and weight constraints [SIM 02]. Thereafter we can fully benefit from the rapid development and increasing performance of SAT solvers in the task of computing stable models.

Acknowledgements

The author wishes to thank anonymous referees for their comments and suggestions for improvement as well as Mirek Truszczyński for the initial idea of applying techniques from [JAN 99a] to the analysis of normal logic programs.

Due to long history, this research has been supported partially by the Academy of Finland (under projects #43963 *"Constraint Programming Based on Default Rules"*, #53695 *"Applications of Rule-Based Constraint Programming"*, and #211025 *"Applications of Constraint Programming Techniques"*) and the European Commission (under contract IST-FET-2001-37004 *"Working Group on Answer Set Programming"*).

10. References

[ANT 01] ANTONIOU G., BILLINGTON D., GOVERNATORI G., MAHER M. J., "Representation Results for Defeasible Logic", *ACM Transactions on Computational Logic*, vol. 2, num. 2, 2001, p. 255-287.

[APT 88] APT K., BLAIR H., WALKER A., "Towards a Theory of Declarative Knowledge", MINKER J., Ed., *Foundations of Deductive Databases and Logic Programming*, p. 89–148, Morgan Kaufmann, Los Altos, 1988.

[ARA 95] ARAVINDAN C., DUNG P. M., "On the Correctness of Unfold/Fold Transformation of Normal and Extended Logic Programs", *Journal of Logic Programming*, vol. 24, num. 3, 1995, p. 201–217.

[BAB 00] BABOVICH Y., ERDEM E., LIFSCHITZ V., "Fages' Theorem and Answer Set Programming", *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*, Breckenridge, Colorado, USA, April 2000, cs.AI/0003042.

[BAY 97] BAYARDO R., SCHRAG R., "Using CSP Look-Back Techniques to Solve Real-World SAT Instances", *Proceedings of the 12th National Conference*, AAAI, 1997, p. 203–208.

[BEN 94] BEN-ELIYAHU R., DECHTER R., "Propositional Semantics for Disjunctive Logic Programs", *Annals of Mathematics and Artificial Intelligence*, vol. 12, num. 1–2, 1994, p. 53–87.

[BRA 97] BRASS S., DIX J., "Characterizations of the Disjunctive Stable Semantics by Partial Evaluation", *Journal of Logic Programming*, vol. 32, num. 3, 1997, p. 207–228.

[BUG 94] BUGLIESI M., LAMMA E., MELLO P., "Modularity in Logic Programming", *Journal of Logic Programming*, vol. 19–20, 1994, p. 443–502.

[CAB 05] CABALAR P., FERRARIS P., "Propositional Theories are Strongly Equivalent to Logic Programs", Submitted for publication, 2005.

[CLA 78] CLARK K. L., "Negation as Failure", GALLAIRE H., MINKER J., Eds., *Logic and Data Bases*, p. 293-322, Plenum Press, New York, 1978.

[COO 71] COOK S. A., "The Complexity of Theorem Proving Procedures", *Proceedings of the third Annual ACM Symposium on Theory of Computing*, 1971, p. 151–158.

[COS 02] COSTANTINI S., D'ANTONA O., PROVETTI A., "On the Equivalence and Range of Applicability of Graph-Based Representations of Logic Programs", *Information Processing Letters*, vol. 84, 2002, p. 241–249.

[DIM 97] DIMOPOULOS Y., NEBEL B., KOEHLER J., "Encoding Planning Problems in Non-monotonic Logic Programs", *Proceedings of the Fourth European Conference on Planning*, Toulouse, France, September 1997, Springer-Verlag, p. 169–181.

[DOW 84] DOWLING W. F., GALLIER J. H., "Linear-Time Algorithms for Testing the Satisfiability of Propositional Horn Formulae", *Journal of Logic Programming*, vol. 3, 1984, p. 267–284.

[EIT 98] EITER T., LEONE N., MATEIS C., PFEIFER G., SCARCELLO F., "The KR System DLV: Progress Report, Comparisons and Benchmarks", *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning*, Trento, Italy, June 1998, Morgan Kaufmann, p. 406–417.

[EIT 04] EITER T., FINK M., TOMPITS H., WOLTRAN S., "Simplifying Logic Programs under Uniform and Strong Equivalence", *Proceedings of LPNMR-7*, Fort Lauderdale, Florida, January 2004, Springer, p. 87–99, LNAI 2923.

[EMD 76] VAN EMDEN M., KOWALSKI R., "The Semantics of predicate logic as a programming language", *Journal of the ACM*, vol. 23, 1976, p. 733-742.

[ERD 01] ERDEM E., LIFSCHITZ V., "Transitive Closure, Answer Sets and Predicate Completion", *AAAI Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, AAAI, 2001.

[ERD 03] ERDEM E., LIFSCHITZ V., "Tight Logic Programs", *Theory and Practice of Logic Programming*, vol. 3, num. 4–5, 2003, p. 499–518.

[ETA 96] ETALLE S., GABBRIELLI M., "Transformations of CLP Modules", *Theoretical Computer Science*, vol. 166, 1996, p. 101–146.

[FAG 94] FAGES F., "Consistency of Clark's Completion and Existence of Stable Models", *Journal of Methods of Logic in Computer Science*, vol. 1, 1994, p. 51-60.

[FER 05] FERRARIS P., "On Modular Translations and Strong Equivalence", BARAL C., GRECO G., LEONE N., TERRACINA G., Eds., *Proceedings of the 8th International Conference on Logic Programming and Nonmonotonic Reasoning*, Diamante, Italy, September 2005, Springer-Verlag, p. 79–91.

[GEL 88] GELFOND M., LIFSCHITZ V., "The Stable Model Semantics for Logic Programming", *Proceedings of the 5th International Conference on Logic Programming*, Seattle, USA, August 1988, The MIT Press, p. 1070–1080.

[GEL 90] GELFOND M., LIFSCHITZ V., "Logic Programs with Classical Negation", *Proceedings of the 7th International Conference on Logic Programming*, Jerusalem, Israel, June 1990, The MIT Press, p. 579–597.

[GEL 91] GELFOND M., LIFSCHITZ V., "Classical Negation in Logic Programs and Disjunctive Databases", *New Generation Computing*, vol. 9, 1991, p. 365–385.

[GEL 02] GELFOND M., LEONE N., "Logic Programming and Knowledge Representation – The A-Prolog Perspective", *Artificial Intelligence*, vol. 138, 2002, p. 3–38.

[GOT 95] GOTTLOB G., "Translating Default Logic into Standard Autoepistemic Logic", *Journal of the ACM*, vol. 42, num. 2, 1995, p. 711-740.

[IMI 87] IMIELINSKI T., "Results on Translating Defaults to Circumscription", *Artificial Intelligence*, vol. 32, 1987, p. 131–146.

[JAN 99a] JANHUNEN T., "Classifying Semi-Normal Default Logic on the Basis of its Expressive Power", GELFOND M., LEONE N., PFEIFER G., Eds., *Proceedings of the 5th International Conference on Logic Programming and Non-Monotonic Reasoning, LPNMR'99*, El Paso, Texas, December 1999, Springer-Verlag, p. 19–33, LNAI 1730.

[JAN 99b] JANHUNEN T., "On the Intertranslatability of Non-monotonic Logics", *Annals of Mathematics and Artificial Intelligence*, vol. 27, num. 1-4, 1999, p. 79–128.

[JAN 00a] JANHUNEN T., "Capturing Stationary and Regular Extensions with Reiter's Extensions", OJEDA-ACIEGO M. et al., Eds., *Logics in Artificial Intelligence, European Workshop, JELIA 2000*, Málaga, Spain, September/October 2000, Springer-Verlag, p. 102–117, LNAI 1919.

[JAN 00b] JANHUNEN T., "Comparing the Expressive Powers of Some Syntactically Restricted Classes of Logic Programs", LLOYD J. et al., Eds., *Computational Logic, First International Conference*, London, UK, July 2000, Springer-Verlag, p. 852–866, LNAI 1861.

[JAN 01] JANHUNEN T., "On the Effect of Default Negation on the Expressiveness of Disjunctive Rules", EITER T., FABER W., TRUSZCZYŃSKI M., Eds., *Logic Programming and Nonmonotonic Reasoning, Proceedings of the 6th International Conference*, Vienna, Austria, September 2001, Springer-Verlag, p. 93–106, LNAI 2173.

[JAN 02] JANHUNEN T., OIKARINEN E., "Testing the Equivalence of Logic Programs under Stable Model Semantics", FLESCA S. et al., Eds., *Logics in Artificial Intelligence, Pro-*

ceedings of the 8th European Conference, Cosenza, Italy, September 2002, Springer-Verlag, p. 493–504, LNAI 2424.

[JAN 03a] JANHUNEN T., "Evaluating the Effect of Semi-Normality on the Expressiveness of Defaults", *Artificial Intelligence*, vol. 144, num. 1–2, 2003, p. 233–250.

[JAN 03b] JANHUNEN T., "Translatability and intranslatability results for certain classes of logic programs", Series A: Research report num. 82, November 2003, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland.

[JAN 04] JANHUNEN T., "Representing Normal Programs with Clauses", DE MÁNTARAS R. L., SAITTA L., Eds., *Proceedings of the 16th European Conference on Artificial Intelligence*, Valencia, Spain, August 2004, IOS Press, p. 358–362.

[KAU 96] KAUTZ H., SELMAN B., "Pushing the Envelope: Planning, Propositional Logic, and Stochastic Search", *Proceedings of the 13th National Conference on Artificial Intelligence*, Portland, Oregon, July 1996.

[LIE 04] LIERLER Y., MARATEA M., "CMODELS-2: SAT-based Answer Set Solver Enhanced to Non-tight Programs", *Proceedings of LPNMR-7*, Fort Lauderdale, Florida, January 2004, Springer, p. 346–350, LNAI 2923.

[LIF 01] LIFSCHITZ V., PEARCE D., VALVERDE A., "Strongly Equivalent Logic Programs", *ACM Transactions on Computational Logic*, vol. 2, 2001, p. 526-541.

[LIF 06] LIFSCHITZ V., RAZBOROV A., "Why Are There So Many Loop Formulas?", *ACM Transactions on Computational Logic*, vol. 7, num. 2, 2006, To appear, see `http://www.acm.org/pubs/tocl/accepted.html`.

[LIN 03] LIN F., ZHAO J., "On Tight Logic Programs and Yet Another Translation from Normal Logic Programs to Propositional Logic", GOTTLOB G., WALSH T., Eds., *the 18th International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, August 2003, Morgan Kaufmann, p. 853–858.

[LIN 04] LIN F., ZHAO Y., "ASSAT: Computing Answer Sets of a Logic Program by SAT solvers", *Artificial Intelligence*, vol. 157, 2004, p. 115–137.

[LLO 87] LLOYD J., *Foundations of Logic Programming*, Springer-Verlag, Berlin, 1987.

[MAH 88] MAHER M. J., "Equivalences of Logic Programs", MINKER J., Ed., *Foundations of Deductive Databases and Logic Programming*, p. 627–658, Morgan Kaufmann, Los Altos, 1988.

[MAH 93] MAHER M. J., "A Tranformation System for Deductive Databases Modules with Perfect Model Semantics", *Theoretical Computer Science*, vol. 110, num. 2, 1993, p. 377–403.

[MAR 91] MAREK W., TRUSZCZYŃSKI M., "Autoepistemic Logic", *Journal of the ACM*, vol. 38, 1991, p. 588–619.

[MAR 92] MAREK V. W., SUBRAHMANIAN V. S., "The Relationship between Stable, Supported, Default and Autoepistemic Semantics for General Logic Programs", *Theoretical Computer Science*, vol. 103, 1992, p. 365–386.

[MAR 99] MAREK W., TRUSZCZYŃSKI M., "Stable Models and an Alternative Logic Programming Paradigm", *The Logic Programming Paradigm: a 25-Year Perspective*, p. 375–398, Springer-Verlag, 1999.

[MOS 01] MOSKEWICZ M., MADIGAN C., ZHAO Y., ZHANG L., MALIK S., "CHAFF: Engineering an Efficient SAT Solver", *Proceedings of the 39th Design Automation Conference*, Las Vegas, 2001.

[NIE 99] NIEMELÄ I., "Logic Programs with Stable Model Semantics as a Constraint Programming Paradigm", *Annals of Mathematics and Artificial Intelligence*, vol. 25, num. 3–4, 1999, p. 241–273.

[NUT 94] NUTE D., "Defeasible Logic", GABBAY D., HOGGER C., ROBINSON J., Eds., *Handbook of Logic in Artificial Intelligence and Logic Programming*, chapter 7, p. 353–395, Oxford Science Publications, 1994.

[PET 94] PETTOROSSI A., PROIETTI M., "Transformation of Logic Programs: Foundations and Techniques", *Journal of Logic Programming*, vol. 19–20, 1994, p. 261–320.

[REI 78] REITER R., "On Closed World Data Bases", GALLAIRE H., MINKER J., Eds., *Logic and Data Bases*, p. 55–76, Plenum Press, New York, 1978.

[ROB 65] ROBINSON J. A., "A Machine-Oriented Logic Based on the Resolution Principle", *Journal of the ACM*, vol. 12, num. 1, 1965, p. 23-41.

[SCH 95] SCHLIPF J., "The Expressive Powers of the Logic Programming Semantics", *Journal of Computer and System Sciences*, vol. 51, 1995, p. 64–86.

[SIM 02] SIMONS P., NIEMELÄ I., SOININEN T., "Extending and Implementing the Stable Model Semantics", *Artificial Intelligence*, vol. 138, num. 1–2, 2002, p. 181–234.

[SUB 95] SUBRAHMANIAN V., NAU D., VAGO C., "WFS + Branch and Bound = Stable Models", *IEEE Transactions on Knowledge and Data Engineering*, vol. 7, num. 3, 1995, p. 362–377.

[TSE 83] TSEITIN G. S., "On the Complexity of Derivation in Propositional Calculus", SIEKMANN J., WRIGHTSON G., Eds., *Automation of Reasoning 2: Classical Papers on Computational Logic 1967-1970*, p. 466-483, Springer, Berlin, Heidelberg, 1983.

[TUR 03] TURNER H., "Strong Equivalence Made Easy: Nested Expressions and Weight Constraints", *Theory and Practice of Logic Programming*, vol. 3, num. 4–5, 2003, p. 609–622.

[VAN 91] VAN GELDER A., ROSS K., SCHLIPF J., "The Well-Founded Semantics for General Logic Programs", *Journal of the ACM*, vol. 38, num. 3, 1991, p. 620–650.

[WOL 04] WOLTRAN S., "Characterizations for Relativized Notions of Equivalence in Answer Set Programming", ALFERES J., LEITE J., Eds., *Logics in Artificial Intelligence: 9th European Conference*, Springer Verlag, September 2004, p. 161–173, LNAI 3229.