A Counter-Based Approach to Translating Normal Logic Programs into Sets of Clauses

Tomi Janhunen

Helsinki University of Technology
Department of Computer Science and Engineering
Laboratory for Theoretical Computer Science
P.O.Box 5400, FIN-02015 HUT, Finland
`Tomi.Janhunen@hut.fi`

Abstract. In this paper, we develop a two-phased translation function from normal logic programs into sets of clauses. The translation is based on a novel characterization of stable models in terms of level numberings and it uses atomic normal programs, which are free of positive body literals, as an intermediary representation. The resulting translation function has attractive properties which are lacked by the earlier attempts. First, a bijective relationship is established between stable models and classical models. Second, the translation can be performed in time proportional to $||P|| \times \log_2 |\mathrm{At}(P)|$ where $||P||$ is the length of a program P in symbols and $|\mathrm{At}(P)|$ is the number of atoms in P.

1 Introduction

Normal logic programs under the stable model semantics [9] are well-suited for a variety of knowledge representation tasks. Typically, a problem at hand is solved (i) by formulating it as a logic program whose stable models correspond to the solutions of the problem and (ii) by computing stable models using a special-purpose search engine. The reader is referred e.g. to [16, 17] for examples of using this kind of methodology, also known as *answer set programming* (ASP).

Similar problems are solvable by formulating them as classical satisfiability (SAT) problems and using SAT solvers. However, such formulations tend to be more difficult and less concise. E.g., formulating an AI planning problem is much easier as a normal logic program [5] than as a set of clauses [13]. This indicates of a real difference in expressive power which can be established formally by showing that normal programs cannot be translated into sets of clauses in a *faithful* and *modular* way [17, 11, 12]. In spite of this intranslatability result, we develop a faithful and *non-modular*, but still fairly systematic, translation function from normal programs into sets of clauses. Using a novel characterization of stable models based on level numberings, the time complexity remains sub-quadratic.

We proceed as follows. In Section 2, we review the syntax and semantics of normal logic programs and sets of clauses. As a further preparatory step, we characterize stable models in terms of level numberings in Section 3. The translation function mentioned above is presented in Section 4. A comparison with related work takes place in Section 5 whereas Section 6 concludes the paper.

2 Preliminaries

A *normal (logic) program* P is a set of expressions or *rules* of the form

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

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. In this paper, we restrict ourselves to the purely propositional case and consider only programs that consist of propositional atoms. The symbol $\sim$ denotes *default negation* or *negation as failure to prove* [4] which differs in an important way from *classical negation* denoted by $\neg$. We define *(default) literals* in the standard way using $\sim$ as the negation sign. Given a rule r of the form (1), the atom a forms the *head* of r whereas the positive literals $\mathsf{b}_1, \ldots, \mathsf{b}_n$ and the negative literals ${\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m$ together form the *body* of r. Despite of the notation used in (1), we interpret the body as a set of literals, which implies that the order of the literals is not relevant. To enable easy reference to the atoms/literals in the body, we adopt the following notations: $\mathrm{H}(r) = \mathsf{a}$, $\mathrm{B}(r) = \{\mathsf{b}_1, \ldots, \mathsf{b}_n\} \cup \{{\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m\}$, $\mathrm{B}^+(r) = \{\mathsf{b}_1, \ldots, \mathsf{b}_n\}$, and $\mathrm{B}^-(r) = \{\mathsf{c}_1, \ldots, \mathsf{c}_m\}$. We generalize these notations for any normal program P in the obvious way: $\mathrm{H}(P) = \{\mathrm{H}(r) \mid r \in P\}$; and $\mathrm{B}(P)$, $\mathrm{B}^+(P)$, and $\mathrm{B}^-(P)$ are defined analogously.

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 literals, i.e. $|\mathrm{B}^+(r)|$, allowed in a rule r [11]. 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. We extend these conditions to cover programs in the obvious way: a logic program P satisfies any of these conditions given that every rule of P satisfies the condition. For instance, an *atomic normal program* P contains only rules of the form $\mathsf{a} \leftarrow {\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m$.

2.1 Semantics

Normal programs can be given a standard model-theoretic semantics. The *Herbrand base* $\mathrm{At}(P)$ of a normal logic program P is defined as the set of atoms that appear in the rules of P. An *interpretation* $I \subseteq \mathrm{At}(P)$ of a normal program P determines which atoms $\mathsf{a} \in \mathrm{At}(P)$ are *true* ($\mathsf{a} \in I$) and which atoms are *false* ($\mathsf{a} \in \mathrm{At}(P) - I$). The satisfaction relation $\models$ is defined for rules and programs in the standard way. Note that negative default literals are given a classical interpretation at this point: $I \models {\sim}\mathsf{a} \iff I \not\models \mathsf{a}$. A rule r is satisfied in I, denoted by $I \models r$, $\iff$ $I \models \mathrm{H}(r)$ is implied by $I \models \mathrm{B}(r)$. Finally, an interpretation I is a *(classical) model* of P, denoted by $I \models P$, $\iff$ $I \models r$ for every $r \in P$.

Although classical models give a semantics for arbitrary normal programs P, the ultimate semantics assigned to normal programs will be different as *minimal models* are distinguished. A model $M \models P$ is a minimal model of P $\iff$ 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 which equals to the intersection of all models of P [14]. 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 P and P', then $\mathrm{LM}(P) \subseteq \mathrm{LM}(P')$.

The least model $\mathrm{LM}(P)$ of a positive program P can be constructed iteratively using the van Emden-Kowalski operator T_P which is defined by $\mathrm{T}_P(A) = \{\mathrm{H}(r) \mid r \in P \text{ and } \mathrm{B}^+(r) \subseteq A\}$ for any set of atoms $A \subseteq \mathrm{At}(P)$. The iteration sequence of T_P is then defined inductively as follows: $\mathrm{T}_P \uparrow 0 = \emptyset$, $\mathrm{T}_P \uparrow i = \mathrm{T}_P(\mathrm{T}_P \uparrow i-1)$ for $i > 0$, and $\mathrm{T}_P \uparrow \omega = \bigcup_{i<\omega} \mathrm{T}_P \uparrow i$. It follows that $\mathrm{LM}(P) = \mathrm{T}_P \uparrow \omega = \mathrm{lfp}(\mathrm{T}_P)$. Note that for finite programs P, $\mathrm{lfp}(\mathrm{T}_P)$ is always reached with a finite number of steps. In the sequel, we use the iterative construction above to define the *level number* $\mathrm{lev}(\mathsf{a})$ for each true atom $\mathsf{a} \in \mathrm{LM}(P)$, i.e. the least natural number i such that $\mathsf{a} \in \mathrm{T}_P \uparrow i$.

Gelfond and Lifschitz [9] propose a way to apply the least model semantics to an arbitrary normal program P. Given an interpretation $M \subseteq \mathrm{At}(P)$, i.e. a *model candidate*, their idea is to reduce P to a positive program $P^M = \{r^+ \mid r \in P \text{ and } M \cap \mathrm{B}^-(r) = \emptyset\}$. In this way, the negative default literals appearing in the bodies of rules 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, which was used to reduce P, leads to the following notion of *stability*.

Definition 1 (Gelfond and Lifschitz [9]). *An interpretation $M \subseteq \mathrm{At}(P)$ of a normal logic program P is a stable model of P $\iff$ $M = \mathrm{LM}(P^M)$.*

Every stable model of P is also a classical model of P, but the converse does not hold necessarily. In general, a normal logic program need not have a unique stable model nor stable models at all. 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 the stable model semantics.

The stable model semantics of normal programs was preceded by an alternative semantics, namely the one based on *supported models* [1]. A classical model M of a normal program P is a supported model of P $\iff$ for every atom $\mathsf{a} \in M$ there is a rule $r \in P$ such that $\mathrm{H}(r) = \mathsf{a}$ and $M \models \mathrm{B}(r)$. As shown in [15], any stable model $M \subseteq \mathrm{At}(P)$ of a normal logic program P is also a supported model of P, but not necessarily vice versa. Supported models can be given a fixed-point characterization in analogy to stable models: an interpretation $M \subseteq \mathrm{At}(P)$ is a supported model of a normal program P $\iff$ $M = \mathrm{T}_{P^M}(M)$. In the sequel, we distinguish 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 $I \subseteq \mathrm{At}(P)$.

2.2 Sets of Clauses

We define *classical literals* in the standard way using classical negation $\neg$ as the connective. A *clause* $C = \{\mathsf{a}_1, \ldots, \mathsf{a}_n, \neg\mathsf{b}_1, \ldots, \neg\mathsf{b}_m\}$ is a finite set of classical literals representing a disjunction of its constituents. A set of clauses S represents a conjunction of the clauses contained in it. The Herbrand base of a set of clauses

S is denoted by $\mathrm{At}(S)$ and interpretations are defined as subsets of $\mathrm{At}(S)$. A clause C of the form above is satisfied in an interpretation $I \iff I \models \mathsf{a}_i$ for some $i \in \{1, \ldots, n\}$ or $I \not\models \mathsf{b}_i$ for some $i \in \{1, \ldots, m\}$. An interpretation $I \subseteq \mathrm{At}(S)$ is a classical model of S, denoted by $I \models S$, $\iff$ each clause $C \in S$ is satisfied in I. A set of clauses S gives rise to a set of classical models $\{M \subseteq \mathrm{At}(S) \mid M \models S\}$. This makes an essential difference with respect to a normal program P for which the set of stable models $\{M \subseteq \mathrm{At}(P) \mid M = \mathrm{LM}(P^M)\}$ is of interest.

3 Yet Another Characterization of Stability

In this section, we develop a characterization of stable models in terms of supported models and *level numberings*, as defined below.

Definition 2. *Let M be a supported model of a normal program P. A function $\# : M \cup \mathrm{SR}(P, M) \to \mathbb{N}$ is a level numbering w.r.t. $M \iff$*

$$\forall \mathsf{a} \in M : \; \#\mathsf{a} = \min\{\#r \mid r \in \mathrm{SR}(P, M) \textit{ and } \mathsf{a} = \mathrm{H}(r)\} \textit{ and} \tag{2}$$

$$\forall r \in \mathrm{SR}(P, M) : \; \#r = \begin{cases} \max\{\#\mathsf{b} \mid \mathsf{b} \in \mathrm{B}^+(r)\} + 1, & \textit{if } \mathrm{B}^+(r) \neq \emptyset. \\ 1, & \textit{otherwise.} \end{cases} \tag{3}$$

Definition 2 can be understood as a generalization of the notion of level numbers, first defined for positive programs in Section 2.1, to the case of normal programs. It is important to realize that a level numbering need not exist for every supported model. This is demonstrated by the following example.

Example 1. 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 an empty domain, since $M_1 \cup \mathrm{SR}(P, M_1) = \emptyset$. For the second, the domain $M_2 \cup \mathrm{SR}(P, M_2) = M_2 \cup P$. The requirements in Definition 2 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 $\#\mathsf{a} = \#\mathsf{a} + 2$, which has no solutions. Hence there is no level numbering w.r.t. M_2. $\square$

Proposition 1. *Let M be a supported model of P. If there is a level numbering $\#$ w.r.t. M, then $\#$ is unique.*

Then the question is how one can determine level numberings in practice. In fact, the scheme introduced for atoms in Section 2.1 can be extended to cover rules as well. Given a *positive program* P, the least model $M = \mathrm{LM}(P)$, and any rule $r \in P$ such that $\mathrm{B}^+(r) = \mathrm{B}(r) \subseteq M$, define the level number

$$\mathrm{lev}(r) = \begin{cases} \max\{\mathrm{lev}(\mathsf{b}) \mid \mathsf{b} \in \mathrm{B}^+(r)\} + 1, & \text{if } \mathrm{B}^+(r) \neq \emptyset. \\ 1, & \text{otherwise.} \end{cases} \tag{4}$$

Assigning level numbers in this way is compatible with Definition 2 which implies a characterization of stable models based on the existence of level numberings.

Theorem 1. *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 is a unique level numbering $\# : M \cup \mathrm{SR}(P, M) \to \mathbb{N}$ w.r.t. M defined as follows.*
   (a) For $\mathsf{a} \in M$, let $\#\mathsf{a} = \mathrm{lev}(\mathsf{a})$.
   (b) 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.*

4 Translating Normal Programs into Sets of Clauses

By our earlier results [11, 12], a *faithful* and *modular* translation function Tr from normal programs into atomic normal programs is impossible, i.e. positive body literals cannot be translated away from rules in a faithful and modular way. More precisely, these results are based on the following properties of translation functions. A translation function Tr is *faithful* if and only if the stable models of a program P and the (stable) models of its translation $\mathrm{Tr}(P)$ are in a bijective relationship and coincide up to $\mathrm{At}(P)$. On the other hand, we define Tr to be *modular* if and only if translations can be formed on a rule-by-rule basis, i.e. $\mathrm{Tr}(P \cup Q) = \mathrm{Tr}(P) \cup \mathrm{Tr}(Q)$ and $\mathrm{Tr}(P) \cap \mathrm{Tr}(Q) = \emptyset$ hold for all disjoint sets of rules P and Q satisfying $P \cap Q = \emptyset$.

A similar impossibility result holds for translations from (atomic) normal programs into sets of clauses under classical models [17, 11, 12]. Despite these intranslatability results, we present a polynomial and faithful translation function $\mathrm{Tr}_{\mathrm{AT}}$ which maps an arbitrary normal program P into an atomic program $\mathrm{Tr}_{\mathrm{AT}}(P)$. Our intranslatability results imply 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 3 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. Let us now recall level numberings from Section 3.

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

As there is no explicit way of representing a level numbering within a normal program, we have to encode such a numbering using propositional atoms. Then a natural solution is to use a binary representation for the individual numbers determined by a level numbering $\#$. Unfortunately, every atom in $\mathrm{At}(P)$ may be assigned a different level number in the worst case. This setting is actually demonstrated in Example 2. Thus the level numbers of atoms may vary from 1 to $|\mathrm{At}(P)|$. Hence the highest possible level number of a rule $r \in P$ is $|\mathrm{At}(P)|+1$, as for r_2 in our example. Although level numbers are positive numbers by definition, we leave room for 0 which is to act as the least binary value. Thus, given a normal program P, we have to be prepared for binary numbers consisting of at most

$$\nabla P = \lceil \log_2(|\mathrm{At}(P)| + 2) \rceil \tag{5}$$

bits. In case of Example 2, we have $\nabla P = 2$ which is enough to represent all the values in the range of the level numbering $\#$ in question. In general, we can establish the following bounds for level numbers in terms of ∇P [12]. If $\# : M \cup \mathrm{SR}(P, M) \to \mathbb{N}$ a level numbering w.r.t. a supported model M of a normal program P, then $0 < \#\mathsf{a} < 2^{\nabla P} - 1$ for each $\mathsf{a} \in M$ and $0 < \#r < 2^{\nabla P}$ for each $r \in \mathrm{SR}(P, M)$. The logarithmic factor embodied in ∇P forms an important design criterion, since would like to keep the length of the translation $||\mathrm{Tr}_{\mathrm{AT}}(P)||$ in symbols as well as the translation time proportional to $||P|| \times \nabla P$ rather than $||P|| \times |\mathrm{At}(P)|$. Hence we strive for a sub-quadratic translation function from normal programs to atomic normal programs. There is potential behind such an objective, since $\nabla P = 14$ for normal programs P with $|\mathrm{At}(P)| = 10000$.

4.1 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 by writing $n[i] = n[i \ldots i]$.

Our idea is to encode the level number $\#\mathsf{a}$ of a particular atom $\mathsf{a} \in \mathrm{At}(P)$ 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. Since atoms may take only two values under the stable model semantics, we equate bits 0 and 1 with the truth values false and true, respectively. Because $\mathrm{Tr}_{\mathrm{AT}}(P)$ is supposed to be an atomic normal program, positive body literals are forbidden and we have to introduce a vector $\overline{\mathsf{a}_1}, \ldots, \overline{\mathsf{a}_j}$ of *complementary* atoms so that we can condition rules on both values of bits. The i^{th} bit of the binary counter associated with a takes the value 0 (resp. 1) $\iff$ 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 2, we need certain primitive operations on binary counters. These primitives will be used as subprograms of the forthcoming translation $\mathrm{Tr}_{\mathrm{AT}}(P)$. The first set of subprograms, as listed in Table 1, concentrates on setting the counters to particular values. 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 assigned to false under stable model semantics. The first subprogram $\mathrm{SEL}_j(\mathsf{a}, \mathsf{c})$ selects a value n in the range $0 \leq n < 2^j$ for the binary counter $\mathsf{a}_1, \ldots, \mathsf{a}_j$ associated with an atom a. The second 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

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

Table 1. Primitives for selecting the values of binary counters

that the latter is the former increased by one (modulo 2^j). The last subprogram $\mathrm{FIX}_j(\mathsf{a},n,\mathsf{c})$ assigns a fixed value $0 \leq n < 2^j$ to the counter associated with a.

In addition to setting the values of counters, we have to be able to compare them. Table 2 lists our basic primitives in this respect. The first subprogram $\mathrm{LT}_j(\mathsf{a},\mathsf{b},\mathsf{c})$ 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 which we associate with a and b. 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 of the comparison. Note that the latter atom captures the greater than or equal relation between the values of the counters in question. The second program $\mathrm{EQ}_j(\mathsf{a},\mathsf{b},\mathsf{c})$ checks if the counters associated with the atoms a and b hold the same value. Only two new atoms $\mathsf{eq}(\mathsf{a},\mathsf{b})$ and $\overline{\mathsf{eq}(\mathsf{a},\mathsf{b})}$, which capture the result of the comparison, are needed.

4.2 Translating Normal Programs into Atomic Ones

We will compose a non-modular translation function $\mathrm{Tr}_{\mathrm{AT}}$ in four steps corresponding to Definitions 3–6 to be presented in the sequel. We will use the program $P = \{\mathsf{a} \leftarrow \mathsf{b};\ \mathsf{b} \leftarrow \mathsf{a}\}$ from Example 1 as our running example and the resulting translation $\mathrm{Tr}_{\mathrm{AT}}(P)$ is specified stepwise in Fig. 1. To achieve faithfulness, one of the aims is to capture each stable model M of a normal logic program P as a stable model N of $\mathrm{Tr}_{\mathrm{AT}}(P)$ which is an atomic program. In the subsequent discussion, M and N are supposed to form a pair of stable models in a one-to-one correspondence, as insisted by faithfulness. The first part of the translation $\mathrm{Tr}_{\mathrm{SUPP}}(P)$ aims to capture a supported model M of P and to define the complementary atom $\overline{\mathsf{a}}$ for each atom $\mathsf{a} \in \mathrm{At}(P)$ appearing in P.

Primitive	Definition
$\mathrm{LT}_j(\mathsf{a},\mathsf{b},\mathsf{c}) =$	$\{\mathsf{lt}(\mathsf{a},\mathsf{b})_i \leftarrow \sim\mathsf{a}_i, \sim\overline{\mathsf{b}_i}, \sim\mathsf{c} \mid 0 < i \leq j\} \cup$
	$\{\mathsf{lt}(\mathsf{a},\mathsf{b})_i \leftarrow \sim\mathsf{a}_i, \sim\mathsf{b}_i, \sim\overline{\mathsf{lt}(\mathsf{a},\mathsf{b})_{i+1}}, \sim\mathsf{c} \mid 0 < i < j\} \cup$
	$\{\mathsf{lt}(\mathsf{a},\mathsf{b})_i \leftarrow \sim\overline{\mathsf{a}_i}, \sim\overline{\mathsf{b}_i}, \sim\overline{\mathsf{lt}(\mathsf{a},\mathsf{b})_{i+1}}, \sim\mathsf{c} \mid 0 < i < j\} \cup$
	$\{\overline{\mathsf{lt}(\mathsf{a},\mathsf{b})_i} \leftarrow \sim\mathsf{lt}(\mathsf{a},\mathsf{b})_i, \sim\mathsf{c} \mid 0 < i \leq j\}$
$\mathrm{EQ}_j(\mathsf{a},\mathsf{b},\mathsf{c}) =$	$\{\overline{\mathsf{eq}(\mathsf{a},\mathsf{b})} \leftarrow \sim\mathsf{a}_i, \sim\overline{\mathsf{b}_i}, \sim\mathsf{c} \mid 0 < i \leq j\} \cup$
	$\{\overline{\mathsf{eq}(\mathsf{a},\mathsf{b})} \leftarrow \sim\overline{\mathsf{a}_i}, \sim\mathsf{b}_i, \sim\mathsf{c} \mid 0 < i \leq j\} \cup$
	$\{\mathsf{eq}(\mathsf{a},\mathsf{b}) \leftarrow \sim\overline{\mathsf{eq}(\mathsf{a},\mathsf{b})}, \sim\mathsf{c}\}$

Table 2. Primitives for comparing the values of binary counters

Definition 3. *For a normal program P, define an atomic normal program*

$$\begin{aligned}\mathrm{Tr}_{\mathrm{SUPP}}(P) = \{&\overline{\mathsf{a}} \leftarrow \sim\mathsf{a} \mid \mathsf{a} \in \mathrm{At}(P)\} \cup \\ &\{\mathsf{bt}(r) \leftarrow \sim\overline{\mathrm{B}^+(r)}, \sim\mathrm{B}^-(r);\ \overline{\mathsf{bt}(r)} \leftarrow \sim\mathsf{bt}(r);\ \mathrm{H}(r) \leftarrow \sim\overline{\mathsf{bt}(r)} \mid r \in P\}. \quad (6)\end{aligned}$$

The other parts of the overall translation will require us to determine when the *body* of a rule $r \in P$ 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. Equation (7) gives the translation $\mathrm{Tr}_{\mathrm{SUPP}}(P)$ for our running example. The next part of the translation introduces counters that are needed to represent a level numbering candidate. Two new atoms $\mathsf{ctr}(\mathsf{a})$ and $\mathsf{nxt}(\mathsf{a})$, which act as names of two counters, are introduced for each atom $\mathsf{a} \in \mathrm{At}(P)$. The eventual purpose of these counters is to hold the values $\#\mathsf{a}$ and $\#\mathsf{a}+1$, respectively, in the binary representation when a belongs to the domain of a level numbering $\#$, i.e. $\mathsf{a} \in M$ (or equivalently, $\overline{\mathsf{a}} \notin N$).

Definition 4. *For a normal program P, define an atomic normal program*

$$\begin{aligned}\mathrm{Tr}_{\mathrm{CTR}}(P) = &\bigcup_{\mathsf{a}\in\mathrm{At}(P)} [\mathrm{SEL}_{\nabla P}(\mathsf{ctr}(\mathsf{a}),\overline{\mathsf{a}}) \cup \mathrm{NXT}_{\nabla P}(\mathsf{ctr}(\mathsf{a}),\mathsf{nxt}(\mathsf{a}),\overline{\mathsf{a}})] \cup \\ &\bigcup_{r\in P \textit{ and } \mathrm{B}^+(r)=\emptyset} \mathrm{FIX}_{\nabla P}(\mathsf{ctr}(r),1,\overline{\mathsf{bt}(r)}) \cup \\ &\bigcup_{r\in P \textit{ and } \mathrm{B}^+(r)\neq\emptyset} \mathrm{SEL}_{\nabla P}(\mathsf{ctr}(r),\overline{\mathsf{bt}(r)}). \quad (11)\end{aligned}$$

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 $\#$,

$$\begin{aligned}&\{\mathsf{a} \leftarrow {\sim}\overline{\mathsf{bt}(r_1)};\ \overline{\mathsf{bt}(r_1)} \leftarrow {\sim}\mathsf{bt}(r_1);\ \mathsf{bt}(r_1) \leftarrow {\sim}\overline{\mathsf{b}};\ \overline{\mathsf{b}} \leftarrow {\sim}\mathsf{b}\}\\ \cup\ &\{\mathsf{b} \leftarrow {\sim}\overline{\mathsf{bt}(r_2)};\ \overline{\mathsf{bt}(r_2)} \leftarrow {\sim}\mathsf{bt}(r_2);\ \mathsf{bt}(r_2) \leftarrow {\sim}\overline{\mathsf{a}};\ \overline{\mathsf{a}} \leftarrow {\sim}\mathsf{a}\}\end{aligned} \tag{7}$$

$$\begin{aligned}&\mathrm{SEL}_2(\mathsf{ctr}(\mathsf{a}),\overline{\mathsf{a}}) \cup \mathrm{NXT}_2(\mathsf{ctr}(\mathsf{a}),\mathsf{nxt}(\mathsf{a}),\overline{\mathsf{a}}) \cup \mathrm{SEL}_2(\mathsf{ctr}(r_1),\overline{\mathsf{bt}(r_1)})\\ \cup\ &\mathrm{SEL}_2(\mathsf{ctr}(\mathsf{b}),\overline{\mathsf{b}}) \cup \mathrm{NXT}_2(\mathsf{ctr}(\mathsf{b}),\mathsf{nxt}(\mathsf{b}),\overline{\mathsf{b}}) \cup \mathrm{SEL}_2(\mathsf{ctr}(r_2),\overline{\mathsf{bt}(r_2)})\end{aligned} \tag{8}$$

$$\begin{aligned}&\mathrm{LT}_2(\mathsf{ctr}(r_1),\mathsf{nxt}(\mathsf{b}),\overline{\mathsf{bt}(r_1)}) \cup \{\mathsf{x} \leftarrow {\sim}\mathsf{x}, {\sim}\overline{\mathsf{bt}(r_1)}, {\sim}\overline{\mathsf{lt}(\mathsf{ctr}(r_1),\mathsf{nxt}(\mathsf{b}))_1}\}\\ \cup\ &\mathrm{LT}_2(\mathsf{ctr}(r_2),\mathsf{nxt}(\mathsf{a}),\overline{\mathsf{bt}(r_2)}) \cup \{\mathsf{x} \leftarrow {\sim}\mathsf{x}, {\sim}\overline{\mathsf{bt}(r_2)}, {\sim}\overline{\mathsf{lt}(\mathsf{ctr}(r_2),\mathsf{nxt}(\mathsf{a}))_1}\}\\ \cup\ &\mathrm{EQ}_2(\mathsf{ctr}(r_1),\mathsf{nxt}(\mathsf{b}),\overline{\mathsf{bt}(r_1)}) \cup \{\mathsf{max}(r_1) \leftarrow {\sim}\overline{\mathsf{bt}(r_1)}, {\sim}\overline{\mathsf{eq}(\mathsf{ctr}(r_1),\mathsf{nxt}(\mathsf{b}))}\}\\ \cup\ &\mathrm{EQ}_2(\mathsf{ctr}(r_2),\mathsf{nxt}(\mathsf{a}),\overline{\mathsf{bt}(r_2)}) \cup \{\mathsf{max}(r_2) \leftarrow {\sim}\overline{\mathsf{bt}(r_2)}, {\sim}\overline{\mathsf{eq}(\mathsf{ctr}(r_2),\mathsf{nxt}(\mathsf{a}))}\}\\ \cup\ &\{\mathsf{x} \leftarrow {\sim}\mathsf{x}, {\sim}\overline{\mathsf{bt}(r_1)}, {\sim}\mathsf{max}(r_1);\ \mathsf{x} \leftarrow {\sim}\mathsf{x}, {\sim}\overline{\mathsf{bt}(r_2)}, {\sim}\mathsf{max}(r_2)\}\end{aligned} \tag{9}$$

$$\begin{aligned}&\mathrm{LT}_2(\mathsf{ctr}(r_1),\mathsf{ctr}(\mathsf{a}),\overline{\mathsf{bt}(r_1)}) \cup \{\mathsf{y} \leftarrow {\sim}\mathsf{y}, {\sim}\overline{\mathsf{bt}(r_1)}, {\sim}\overline{\mathsf{lt}(\mathsf{ctr}(r_1),\mathsf{ctr}(\mathsf{a}))_1}\}\\ \cup\ &\mathrm{LT}_2(\mathsf{ctr}(r_2),\mathsf{ctr}(\mathsf{b}),\overline{\mathsf{bt}(r_2)}) \cup \{\mathsf{y} \leftarrow {\sim}\mathsf{y}, {\sim}\overline{\mathsf{bt}(r_2)}, {\sim}\overline{\mathsf{lt}(\mathsf{ctr}(r_2),\mathsf{ctr}(\mathsf{b}))_1}\}\\ \cup\ &\mathrm{EQ}_2(\mathsf{ctr}(r_1),\mathsf{ctr}(\mathsf{a}),\overline{\mathsf{bt}(r_1)}) \cup \{\mathsf{min}(\mathsf{a}) \leftarrow {\sim}\overline{\mathsf{bt}(r_1)}, {\sim}\overline{\mathsf{eq}(\mathsf{ctr}(r_1),\mathsf{ctr}(\mathsf{a}))}\}\\ \cup\ &\mathrm{EQ}_2(\mathsf{ctr}(r_2),\mathsf{ctr}(\mathsf{b}),\overline{\mathsf{bt}(r_2)}) \cup \{\mathsf{min}(\mathsf{b}) \leftarrow {\sim}\overline{\mathsf{bt}(r_2)}, {\sim}\overline{\mathsf{eq}(\mathsf{ctr}(r_2),\mathsf{ctr}(\mathsf{b}))}\}\\ \cup\ &\{\mathsf{y} \leftarrow {\sim}\mathsf{y}, {\sim}\overline{\mathsf{a}}, {\sim}\mathsf{min}(\mathsf{a});\ \mathsf{y} \leftarrow {\sim}\mathsf{y}, {\sim}\overline{\mathsf{b}}, {\sim}\mathsf{min}(\mathsf{b})\}.\end{aligned} \tag{10}$$

Fig. 1. The translation $\mathrm{Tr}_{\mathrm{AT}}(P)$ for the program P given in Example 1

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, which is in perfect accordance with Definition 2. The subprograms which are needed in case of our running example are listed in (8).

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 2. 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 in the forthcoming translation. The case of atomic rules $r \in P$ is already covered by $\mathrm{Tr}_{\mathrm{CTR}}(P)$, but for non-atomic rules $r \in P$ with $\mathrm{B}^+(r) \neq \emptyset$, the maximization principle from Definition 2 must be expressed e.g. as follows.

Definition 5. *Let* x *be a new atom not appearing in* $\mathrm{At}(P)$. *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* $\mathrm{Tr}_{\mathrm{MAX}}(r, b, \mathsf{a}) =$

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

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

$$\begin{aligned}\mathrm{Tr}_{\mathrm{MAX}}(P) = &\bigcup_{r \in P \text{ and } \mathrm{B}^+(r) \neq \emptyset} \mathrm{Tr}_{\mathrm{MAX}}(r, \nabla P) \cup\\ &\{\mathsf{x} \leftarrow {\sim}\mathsf{x}, {\sim}\overline{\mathsf{bt}(r)}, {\sim}\mathsf{max}(r) \mid r \in P \text{ and } \mathrm{B}^+(r) \neq \emptyset\}.\end{aligned} \tag{12}$$

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 literal $\mathsf{a} \in \mathrm{B}^{+}(r)$. 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 perfectly to the definition of $\#r$ given in Definition 2. See (9) in Fig. 1 for the rules involved in the translation $\mathrm{Tr}_{\mathrm{MAX}}(P)$ for our running example.

Let us then turn our attention to atoms a that are assigned to true in a supported model M of P. The properties of supported models imply that there must be 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 2.

Definition 6. *Let* y *be a new atom not appearing in* $\mathrm{At}(P)$. *For a rule* r *and a number of bits* b, *define* $\mathrm{Tr}_{\mathrm{MIN}}(r, b) =$

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

where $\mathsf{a} = \mathrm{H}(r)$. *For a normal program* P, *define an atomic normal program*

$$\mathrm{Tr}_{\mathrm{MIN}}(P) = \bigcup_{r \in P} \mathrm{Tr}_{\mathrm{MIN}}(r, \nabla P) \ \cup \{\mathsf{y} \leftarrow {\sim}\mathsf{y}, {\sim}\overline{\mathsf{a}}, {\sim}\mathsf{min}(\mathsf{a}) \mid \mathsf{a} \in \mathrm{At}(P)\}. \quad (13)$$

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 as dictated by the definition of $\#\mathsf{a}$ in Definition 2. For our running example, the translation $\mathrm{Tr}_{\mathrm{MIN}}(P)$ appears as (10).

We are now ready to formulate the translation function $\mathrm{Tr}_{\mathrm{AT}}$ based on the four sub-translations presented so far. Given a normal program P, we define an atomic normal program $\mathrm{Tr}_{\mathrm{AT}}(P)$ as the union $\mathrm{Tr}_{\mathrm{SUPP}}(P) \cup \mathrm{Tr}_{\mathrm{CTR}}(P) \cup \mathrm{Tr}_{\mathrm{MAX}}(P) \cup \mathrm{Tr}_{\mathrm{MIN}}(P)$. Despite non-modularity, the translation $\mathrm{Tr}_{\mathrm{AT}}(P)$ can be formed in a very systematic fashion by generating certain rules for each $r \in P$ and each $\mathsf{a} \in \mathrm{At}(P)$. A source of non-modularity is hidden in the number of bits ∇P involved in $\mathrm{Tr}_{\mathrm{AT}}(P)$. Given two disjoint programs P and Q, it is possible that $\nabla P < \nabla(P \cup Q)$ and $\nabla Q < \nabla(P \cup Q)$. As a consequence, the counters in $\mathrm{Tr}_{\mathrm{AT}}(P)$ and $\mathrm{Tr}_{\mathrm{AT}}(Q)$ are likely to have too few bits so that $\mathrm{Tr}_{\mathrm{AT}}(P)$ and $\mathrm{Tr}_{\mathrm{AT}}(Q)$ cannot be joined together in order to form the translation $\mathrm{Tr}_{\mathrm{AT}}(P \cup Q)$.

4.3 Correctness of the Translation Function $\mathrm{Tr}_{\mathrm{AT}}$

Our next goal is to specify the expected outcomes of the primitives listed in Tables 1 and 2. If c is cannot be inferred, the contribution of a subprogram

$$\begin{aligned}
\mathrm{Ext}_{\mathrm{SUPP}}(P,M) &= M\cup\{\overline{\mathsf{a}}\mid \mathsf{a}\in\mathrm{At}(P)-M\}\cup\\
&\quad \{\mathsf{bt}(r)\mid r\in\mathrm{SR}(P,M)\}\cup\{\overline{\mathsf{bt}(r)}\mid r\in P-\mathrm{SR}(P,M)\}.\\
\mathrm{Ext}_{\mathrm{CTR}}(P,M,\#) &= \textstyle\bigcup_{\mathsf{a}\in M}\mathrm{AT}^{\mathrm{ctr}}_{\nabla P}(\mathsf{ctr}(\mathsf{a}),\#\mathsf{a})\cup\\
&\quad \textstyle\bigcup_{\mathsf{a}\in M}\mathrm{AT}^{\mathrm{ctr}}_{\nabla P}(\mathsf{nxt}(\mathsf{a}),\#\mathsf{a}+1 \bmod 2^{\nabla P})]\cup\\
&\quad \textstyle\bigcup_{r\in\mathrm{SR}(P,M),\ \mathrm{B}^{+}(r)=\emptyset}\mathrm{AT}^{\mathrm{ctr}}_{\nabla P}(\mathsf{ctr}(r),1)\cup\\
&\quad \textstyle\bigcup_{r\in\mathrm{SR}(P,M),\ \mathrm{B}^{+}(r)\neq\emptyset}\mathrm{AT}^{\mathrm{ctr}}_{\nabla P}(\mathsf{ctr}(r),\#r).\\
\mathrm{Ext}_{\mathrm{MAX}}(P,M,\#) &= \{\mathsf{max}(r)\mid r\in\mathrm{SR}(P,M)\text{ and }\mathrm{B}^{+}(r)\neq\emptyset\}\cup\\
&\quad \textstyle\bigcup_{r\in\mathrm{SR}(P,M),\ \mathsf{a}\in\mathrm{B}^{+}(r)}\mathrm{AT}^{\mathrm{lt}}_{\nabla P}(\mathsf{ctr}(r),\#r,\mathsf{nxt}(\mathsf{a}),\#\mathsf{a}+1 \bmod 2^{\nabla P})\cup\\
&\quad \textstyle\bigcup_{r\in\mathrm{SR}(P,M),\ \mathsf{a}\in\mathrm{B}^{+}(r)}\mathrm{AT}^{\mathrm{eq}}_{\nabla P}(\mathsf{ctr}(r),\#r,\mathsf{nxt}(\mathsf{a}),\#\mathsf{a}+1 \bmod 2^{\nabla P}).\\
\mathrm{Ext}_{\mathrm{MIN}}(P,M,\#) &= \{\mathsf{min}(\mathsf{a})\mid \mathsf{a}\in M\}\cup\\
&\quad \textstyle\bigcup_{r\in\mathrm{SR}(P,M)}\mathrm{AT}^{\mathrm{lt}}_{\nabla P}(\mathsf{ctr}(r),\#r,\mathsf{ctr}(\mathrm{H}(r)),\#\mathrm{H}(r))\cup\\
&\quad \textstyle\bigcup_{r\in\mathrm{SR}(P,M)}\mathrm{AT}^{\mathrm{eq}}_{\nabla P}(\mathsf{ctr}(r),\#r,\mathsf{ctr}(\mathrm{H}(r)),\#\mathrm{H}(r)).
\end{aligned}$$

Fig. 2. Operators for extending a stable model M of P to one of $\mathrm{Tr}_{\mathrm{AT}}(P)$.

$\mathrm{SEL}_j(\mathsf{a},\mathsf{c})$ is a set of true atoms $\mathrm{AT}^{\mathrm{ctr}}_j(\mathsf{a},n)$ which contains for each $0<i\leq j$, the atom $\mathsf{a}_i \iff n[i]=1$, and the atom $\overline{\mathsf{a}_i} \iff n[i]=0$. Here j is the number of bits and n is the value $0\leq n<2^j$ *chosen* for the counter associated with a. If the counter associated with an another atom b is holding a value m such that $m+1=n$ holds modulo 2^j, the same set of atoms is made true by the subprogram $\mathrm{NXT}_j(\mathsf{b},\mathsf{a},\mathsf{c})$, if c is not inferable. Similarly, the set of true atoms $\mathrm{AT}^{\mathrm{ctr}}_j(\mathsf{a},n)$ is obtained with $\mathrm{FIX}_j(\mathsf{a},n,\mathsf{c})$ which assigns a fixed value n to the counter associated with a. Let us then define the outcome of the subprogram $\mathrm{LT}_j(\mathsf{a},\mathsf{b},\mathsf{c})$ when the atom c is false. Given the values $0\leq n<2^j$ and $0\leq m<2^j$ of the counters associated with a and b, respectively, the set of true atoms is $\mathrm{AT}^{\mathrm{lt}}_j(\mathsf{a},n,\mathsf{b},m)$, which contains for each $0<i\leq j$, the atom $\mathsf{lt}(\mathsf{a},\mathsf{b})_i \iff n[i\ldots j]<m[i\ldots j]$, and the atom $\overline{\mathsf{lt}(\mathsf{a},\mathsf{b})_i} \iff n[i\ldots j]\geq m[i\ldots j]$. The program $\mathrm{EQ}_j(\mathsf{a},\mathsf{b},\mathsf{c})$ is covered as follows. If c cannot be inferred, then the set of true atoms $\mathrm{AT}^{\mathrm{eq}}_j(\mathsf{a},n,\mathsf{b},m)$ is $\{\mathsf{eq}(\mathsf{a},\mathsf{b})\}$, if $n=m$, and $\{\overline{\mathsf{eq}(\mathsf{a},\mathsf{b})}\}$, if $n\neq m$.

We are now ready to address the correctness of the translation function $\mathrm{Tr}_{\mathrm{AT}}$. Given a normal program P, an interpretation $M\subseteq\mathrm{At}(P)$ of P, and a function $\#: M\cup\mathrm{SR}(P,M)\rightarrow\{0,\ldots,2^{\nabla P}-1\}$, we write $\mathrm{Ext}_{\mathrm{AT}}(P,M,\#)$ for the union of the sets atoms given in Fig. 2. Moreover, given an interpretation N of the translation $\mathrm{Tr}_{\mathrm{AT}}(P)$, we may *extract* the value of a counter associated with an atom a by setting $\mathrm{val}_j(\mathsf{a},N)=\sum\{2^{j-i}\mid 0<i\leq j\text{ and }\mathsf{a}_i\in N\}$.

Proposition 2. *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 $M=N\cap\mathrm{At}(P)$.*

For the program P given in Example 1, the only stable model $M=\emptyset$ of P is captured as a stable model $N=\{\overline{\mathsf{a}},\overline{\mathsf{b}},\overline{\mathsf{bt}(r_1)},\overline{\mathsf{bt}(r_2)}\}$ of the translation in Fig. 1.

Definition 7. *Let P be a normal program, $N\subseteq\mathrm{At}(\mathrm{Tr}_{\mathrm{AT}}(P))$ an interpretation of the translation $\mathrm{Tr}_{\mathrm{AT}}(P)$, and $M=N\cap\mathrm{At}(P)$. Define a function*

$\# : M \cup \mathrm{SR}(P, M) \to \{0, \ldots, 2^{\nabla P} - 1\}$ *by setting (i)* $\#\mathsf{a} = \mathrm{val}_{\nabla P}(\mathsf{ctr}(\mathsf{a}), N)$ *for atoms* $\mathsf{a} \in M$*, and (ii)* $\#r = \mathrm{val}_{\nabla P}(\mathsf{ctr}(r), N)$ *for rules* $r \in \mathrm{SR}(P, M)$*.*

Proposition 3. *Let P be a normal program. If N is a stable model of the translation* $\mathrm{Tr}_{\mathrm{AT}}(P)$*, then* $M = N \cap \mathrm{At}(P)$ *is a stable model of P and* $N = \mathrm{Ext}_{\mathrm{AT}}(P, M, \#)$ *where $\#$ is the level numbering extracted in Definition 7.*

As a consequence of Propositions 2 and 3, the stable models of a normal program P and the translation $\mathrm{Tr}_{\mathrm{AT}}(P)$ are in a bijective relationship and the models coincide up to $\mathrm{At}(P)$. Thus $\mathrm{Tr}_{\mathrm{AT}}$ is faithful in the sense explained in the beginning of Section 4. Furthermore, it can be established that $\mathrm{Tr}_{\mathrm{AT}}(P)$ can be produced in time linear to $||P|| \times \log_2 |\mathrm{At}(P)|$, see [12] for details.

4.4 Translating Atomic Normal Programs into Sets of Clauses

Atomic normal programs provide a promising intermediary representation that is relatively straightforward to translate into a set of propositional clauses. Such programs are *positive order consistent* in the sense proposed by Fages [8]. As a consequence, stable and supported models coincide for this class of programs, and Clark's program completion is sufficient to capture stable models. However, new atoms have to be introduced in order to keep the translation function linear.

Definition 8. *For an atomic normal program P and an atom* $\mathsf{a} \in \mathrm{At}(P)$*, let* $\mathrm{Def}_P(\mathsf{a}) = \{r \in P \mid \mathrm{H}(r) = \mathsf{a}\}$ *and define the set of clauses*

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

where $\mathsf{bt}(r)$ *is a new atom for each* $r \in P$ *and* $\mathrm{Tr}_{\mathrm{CL}}(P) = \bigcup_{\mathsf{a} \in \mathrm{At}(P)} \mathrm{Tr}_{\mathrm{CL}}(\mathsf{a}, P)$*.*

The intuitive reading of $\mathsf{bt}(r)$ is the same as in $\mathrm{Tr}_{\mathrm{AT}}$. Roughly speaking, the clauses in the translation ensure that every atom $\mathsf{a} \in \mathrm{At}(P)$ is logically equivalent to the disjunction of all bodies of rules $r \in P$ with $\mathrm{H}(r) = \mathsf{a}$. This leads to a tight (bijective) correspondence of models as described next. Given an interpretation $I \subseteq \mathrm{At}(P)$ of a program P, define $\mathrm{Ext}_{\mathrm{CL}}(P, I) = I \cup \{\mathsf{bt}(r) \mid r \in \mathrm{SR}(P, I)\}$.

Proposition 4. *Let P be an atomic normal program. If* $M \subseteq \mathrm{At}(P)$ *is a stable model of P, then* $N = \mathrm{Ext}_{\mathrm{CL}}(P, M)$ *is a model of* $\mathrm{Tr}_{\mathrm{CL}}(P)$ *such that* $M = N \cap \mathrm{At}(P)$*. If an interpretation* $N \subseteq \mathrm{At}(\mathrm{Tr}_{\mathrm{CL}}(P))$ *is a (classical) model* $\mathrm{Tr}_{\mathrm{CL}}(P)$*, then* $M = N \cap \mathrm{At}(P)$ *is a stable model of P such that* $N = \mathrm{Ext}_{\mathrm{CL}}(P, M)$*.*

The translation function $\mathrm{Tr}_{\mathrm{CL}}$ is clearly non-modular, as 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 atom a. Thus $\mathrm{Tr}_{\mathrm{CL}}(P)$ cannot be formed on a rule-by-rule basis. On the other hand, the composition $\mathrm{Tr}_{\mathrm{AT}} \circ \mathrm{Tr}_{\mathrm{CL}}$ of the two translation functions maps an arbitrary normal program P into a set of clauses $\mathrm{Tr}_{\mathrm{CL}}(\mathrm{Tr}_{\mathrm{AT}}(P))$ such that the stable models of P and the classical models of the translation are in a bijective relationship and coincide up to $\mathrm{At}(P)$. Moreover, the translation can be formed in time linear to $||P|| \times \log_2 |\mathrm{At}(P)|$.

5 Related Work

Ben-Eliyahu and Dechter [3] study the possibilities of reducing *head-cycle-free* disjunctive logic programs, under the stable model semantics [10], to propositional logic. As normal programs form a special case of head-cycle-free disjunctive programs, a comparison with our results follows. One of the results obtained by Ben-Eliyahu and Dechter [3, Theorem 2.8] is a characterization of stable models that resembles the one developed in Section 3. However, they impose weaker conditions on level numberings. That is, they insist on the existence of a function $f : \mathrm{At}(P) \rightarrow \mathbb{N}^{+}$ such that for each $\mathsf{a} \in M$, there is a rule $r \in \mathrm{SR}(P, M)$ satisfying $f(\mathsf{b}) < f(\mathsf{a})$ for every $\mathsf{b} \in \mathrm{B}^{+}(r)$. It is easy to see that a level numbering $\#$ conforming to Definition 2 can be extended to such a function f, but such functions are by no means unique even if the range of f is limited. This is in contrast to Theorem 1 where the uniqueness of level numberings is established.

The translation function $\mathrm{Tr}_{\mathrm{BD}}$ (called *translate-2* in [3]) proposed by Ben-Eliyahu and Dechter produces a propositional theory $\mathrm{Tr}_{\mathrm{BD}}(P)$ that consists of four parts. The first two parts ensure that each model N of $\mathrm{Tr}_{\mathrm{BD}}(P)$ captures a classical model M of P. The third part makes M a supported and stable model of P whereas the fourth part can be neglected in case of normal programs. In particular, the fact that $f(\mathsf{a}) = i$ holds for an atom $\mathsf{a} \in \mathrm{At}(P)$ is expressed by making a new atom $\mathsf{in}(\mathsf{a})_i$ true in N. Similar objectives can be identified for the sub-translations involved in $\mathrm{Tr}_{\mathrm{AT}}(P)$. 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 one-to-one correspondence between the stable models of P and the classical models of the translation. This is because the level numberings used by Ben-Eliyahu and Dechter are not unique. Moreover, the language of P is not preserved by the translation function $\mathrm{Tr}_{\mathrm{BD}}$, as $\mathrm{At}(P) \cap \mathrm{At}(\mathrm{Tr}_{\mathrm{BD}}(P)) = \emptyset$. A further difference is that $||\mathrm{Tr}_{\mathrm{BD}}(P)||$ is quadratic in $||P||$ in the worst case. The translation function $\mathrm{Tr}_{\mathrm{AT}} \circ \mathrm{Tr}_{\mathrm{CL}}$ is more compact, as a binary encoding of level numbers is used.

There are also other characterizations of stable models that are closely related to the one established in Section 3. Fages [8] calls an interpretation $I \subseteq \mathrm{At}(P)$ 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 is $r \in \mathrm{SR}(P, I)$ satisfying $\mathrm{H}(r) = \mathsf{a}$ and $\mathsf{b} \prec \mathsf{a}$ for all $\mathsf{b} \in \mathrm{B}^{+}(r)$. The basic result [8, Theorem 3.2] that well-supported models of a normal program P are stable models of P, and vice versa. In fact, it is possible to associate such an ordering with a level numbering conforming to Definition 2: just define $\mathsf{a} \prec \mathsf{b} \iff \#\mathsf{a} < \#\mathsf{b}$ for any $\mathsf{a} \in I$ and $\mathsf{b} \in I$. The resulting ordering can be considered as a canonical one, as $\#$ is known to be unique by Theorem 1. Moreover, Fages distinguishes ***positive order consistent*** normal programs whose models are necessarily well-supported. As a consequence, the classical models of the completed program P [4], or supported models of P, coincide with the stable models of P.

Quite recently, Babovich et al. [2] and also Erdem and Lifschitz [7] generalize Fages' results by introducing the notion of *tightness* for logic programs. The tightness of a logic program P is defined relative a set atoms $A \subseteq \mathrm{At}(P)$, which makes Fages' theorem applicable to a wider range of programs. To understand the contribution of this paper in this respect, let us point out that atomic normal programs are automatically positive order consistent, or *absolutely* tight in the terminology of [7]. Therefore, arbitrary normal programs can be transformed into absolutely tight ones in a fairly systematic fashion by applying the translation function $\mathrm{Tr}_{\mathrm{AT}}$ presented in Section 4. A further implication is that a *transitive closure* of relation can be properly captured with classical models. This has already been established by Erdem and Lifschitz [6] for relations that can be represented in terms of a tight program, but our results generalize this fact for arbitrary logic programs. There are also problems, such as Hamiltonian cycles in graphs, for which no representations as tight programs are known. In this case, a representation is obtained by applying $\mathrm{Tr}_{\mathrm{AT}}$ to a formulation by Niemelä [17].

6 Conclusions

In this paper, we tackle a very challenging problem of translating normal programs into sets of clauses so that a one-to-one correspondence of models is obtained. The results of the paper indicate that such a transformation is possible and of reasonably low time complexity, although it cannot be done rule-by-rule. The reader is referred to [12] for full proofs and further (in)translatability results.

The characterization of stable models developed in Section 3 reveals that the computation of the least model for a positive normal program can be viewed as a minimization/maximization process. As discussed in Section 5, a particular novelty of a level numbering conforming to Definition 2 is that the values assigned to atoms are uniquely determined. This is in sharp contrast with earlier characterizations of stable models, where similar numberings are used to distinguish stable models, but the value assignment can be done even in infinitely many ways. Unique level numberings are crucial for the main objective of Section 4, i.e. obtaining a tight (bijective) correspondence between models.

In Section 4, we develop a counter-based approach for translating normal programs into atomic ones. Compared to earlier attempts, there are several distinctive features in our approach. As a fundamental result, all finite normal programs can be covered and a bijective relationship of models is obtained. Moreover, the translation function $\mathrm{Tr}_{\mathrm{AT}}$ preserves the Herbrand base of the program, only new atoms are added. The length of the translation $||\mathrm{Tr}_{\mathrm{AT}}(P)||$ as well as the translation time are of order $||P|| \times \log_2 |\mathrm{At}(P)|$, indicating that $\mathrm{Tr}_{\mathrm{AT}}$ is sub-quadratic. We consider this as a breakthrough, since the best known transformation to date [3] is quadratic. However, the translation function $\mathrm{Tr}_{\mathrm{AT}}$ is far from being optimal, as the translation of two rules given in Fig. 1 consists already of tens of rules. There are several techniques that can be used to decrease the number of rules that have to be generated for a particular normal program, the number of binary counters as well as the number of bits involved in them. One particular technique is to apply $\mathrm{Tr}_{\mathrm{AT}}$ to the *strongly connected components* of P, as already suggested by Ben-Eliyahu and Dechter [3]. We leave such optimizations as implementation issues to be addressed elsewhere, as the main interest in the current paper is to establish a translation function possessing promising properties.

Acknowledgements This work has been supported by the Academy of Finland under Project #53695 "Applications of Rule-Based Constraint Programming". The author wishes to thank the anonymous referees for their comments.

References

1. K.R. Apt, H.A. Blair, and A. Walker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann Publishers, Los Altos, 1988.
2. Y. Babovich, E. Erdem, and V. Lifschitz. Fages' theorem and answer set programming. In *Proceedings of the 8th International Workshop on Non-Monotonic Reasoning*, Breckenridge, Colorado, USA, April 2000. cs.AI/0003042.
3. R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Mathematics and Artificial Intelligence*, 12(1–2):53–87, 1994.
4. K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.
5. Y. Dimopoulos, B. Nebel, and J. Koehler. Encoding planning problems in non-monotonic logic programs. In *Proceedings of the Fourth European Conference on Planning*, pages 169–181, Toulouse, France, September 1997. Springer-Verlag.
6. E. Erdem and V. Lifschitz. Transitive closure, answer sets and predicate completion. In *AAAI Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*. AAAI, 2001.
7. E. Erdem and V. Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming*, 3(4–5):499–518, 2003.
8. F. Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.
9. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proceedings of the 5th International Conference on Logic Programming*, pages 1070–1080, Seattle, USA, August 1988. The MIT Press.
10. M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.
11. T. Janhunen. Comparing the expressive powers of some syntactically restricted classes of logic programs. In *Computational Logic, First International Conference*, pages 852–866, London, UK, July 2000. Springer-Verlag. LNAI 1861.
12. T. Janhunen. Translatability and intranslatability results for certain classes of logic programs. Series A: Research reports, Helsinki University of Technology, Laboratory for Theoretical Computer Science, 2003. To appear.
13. H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the 13th National Conference on Artificial Intelligence*, Portland, Oregon, July 1996.
14. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, Berlin, 1987.
15. V.W. Marek and V.S. Subrahmanian. The relationship between stable, supported, default and autoepistemic semantics for general logic programs. *Theoretical Computer Science*, 103:365–386, 1992.
16. W. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, pages 375–398. Springer-Verlag, 1999.
17. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.