Representing Normal Programs with Clauses

Tomi Janhunen[1]

Abstract. We present a new method for transforming normal logic programs into sets of clauses. This transformation 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 atoms, as an intermediary representation. The corresponding translation function possesses a unique combination of properties: (i) a bijective relationship is established between stable models and classical models, (ii) the models coincide up to the set of atoms $\mathrm{At}(P)$ appearing in a program P, and (iii) the length of the translation as well as the translation time are of order $||P|| \times \log_2 |\mathrm{At}(P)|$ where $||P||$ is the length of the program P. Our preliminary experiments with an implementation of the transformation, namely translators called LP2ATOMIC and LP2SAT, and SAT solvers such as CHAFF and RELSAT suggest that our approach is competitive when the task is to compute not just one but all stable models for a normal program given as input.

1 INTRODUCTION

Normal logic programs under the stable model semantics [9] 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 using a special-purpose search engine. The reader is referred e.g. to [18, 20] 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 [6] than as a set of clauses [12]. 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 [20, 10, 11]. In spite of these intranslatability results, we develop a faithful and *non-modular*, but still fairly systematic, translation from normal programs into sets of clauses. Using a novel characterization of stable models based on level numberings, the time complexity remains sub-quadratic.

The rest of this article is structured as follows. In Section 2, we review the syntax and semantics of normal logic programs as well as sets of clauses. As a further preparatory step, we characterize stable models in terms of level numberings in Section 3. The translation mentioned above is presented in Section 4. We report on our preliminary experiments in Section 5. The discussion in Section 6 concludes the paper.

[1] Helsinki University of Technology, Espoo, Otaniemi, Finland email: Tomi.Janhunen@hut.fi

2 PRELIMINARIES

In this paper, we restrict ourselves to the purely propositional case and consider only programs that consist of propositional atoms. A *normal (logic) program* P is a set of *rules* which are expressions 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. Here $\sim$ denotes *default negation* or Clark's *negation as failure to prove*, which differs from *classical negation* denoted by $\neg$. Intuitively speaking, a rule r of the form (1) is used as follows: if 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 $\mathrm{B}^-(r) = \{\mathsf{c}_1, \ldots, \mathsf{c}_m\}$, then the *head atom* $\mathrm{H}(r) = \mathsf{a}$ can be inferred by applying r.

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, i.e. $|\mathrm{B}^+(r)|$, allowed in a rule r. 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 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 $\mathsf{a} \leftarrow {\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m$.

Let us then turn our attention to the semantics of normal programs. We write $\mathrm{At}(P)$ for the set of atoms that appear in a program P. An *interpretation* $I \subseteq \mathrm{At}(P)$ of P determines which atoms $\mathsf{a} \in \mathrm{At}(P)$ are *true* ($\mathsf{a} \in I$) and which atoms are *false* ($\mathsf{a} \in \mathrm{At}(P) - I$). A rule r is satisfied in I, denoted by $I \models r$, iff $I \models \mathrm{H}(r)$ is implied by $I \models \mathrm{B}(r)$ where $\mathrm{B}(r) = \{\mathsf{b}_1, \ldots, \mathsf{b}_n\} \cup \{{\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m\}$ and $\sim$ is interpreted classically, i.e. $I \models {\sim}\mathsf{c}_i$ iff $I \not\models \mathsf{c}_i$. Then an interpretation I is a *(classical) model* of P, denoted $I \models P$, iff $I \models r$ for each $r \in P$.

But the semantics of normal programs is not solely based on classical models. 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 normal program P has a unique minimal model which equals to the intersection of all models of P [16]. 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 $\mathrm{LM}(P) \subseteq \mathrm{LM}(P')$. Gelfond and Lifschitz [9] extend the least model semantics for arbitrary normal programs. Given a *model candidate* $M \subseteq \mathrm{At}(P)$, the idea is to reduce P to a positive program $P^M = \{r^+ \mid r \in P \text{ and } M \cap \mathrm{B}^-(r) = \emptyset\}$ having the least model $\mathrm{LM}(P^M)$. Equating this model with the model candidate M implies the definition of a *stable model* [9]: $M = \mathrm{LM}(P^M)$ which implies $M \models P$, but not vice versa.

In general, a normal logic program need not have a unique stable model nor stable models at all. 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 $M \models \mathrm{B}(r)$. Inspired by this idea, we define for any program P and $I \subseteq \mathrm{At}(P)$, the set of *supporting rules* $\mathrm{SR}(P, I) = \{r \in P \mid I \models \mathrm{B}(r)\} \subseteq P$. As shown in [17], any stable model $M \subseteq \mathrm{At}(P)$ of a normal logic program P is also a supported model of P, but not vice versa in general.

We define *classical literals* in the standard way using classical negation $\neg$ as the connective. Syntactically, 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. We define the set of atoms $\mathrm{At}(S)$ and interpretations $I \subseteq \mathrm{At}(S)$ in analogy to normal programs. 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. Finally, a set of clauses S gives rise to a set of classical models $\mathrm{CM}(S) = \{M \subseteq \mathrm{At}(S) \mid M \models S\}$. This differs essentially from a normal program P for which the set of stable models $\mathrm{SM}(P) = \{M \subseteq \mathrm{At}(P) \mid M = \mathrm{LM}(P^M)\}$ is of interest.

3 CHARACTERIZING STABILITY

In this section, we characterize stable models in terms of supported models and *level numberings* to be defined next.

Definition 1 *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 for all $\mathsf{a} \in M$,*

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

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

$$\#r = \max\{\#\mathsf{b} \mid \mathsf{b} \in \mathrm{B}^+(r)\} + 1 \tag{3}$$

where we interpret $\max \emptyset$ as 0 to cover rules r with $\mathrm{B}^+(r) = \emptyset$.

It is important to realize that a level numbering need not exist for every supported model, as demonstrated below.

Example 2 *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 1 lead to four equations: $\#\mathsf{a} = \#r_1$, $\#r_1 = \#\mathsf{b} + 1$, $\#\mathsf{b} = \#r_2$, and $\#r_2 = \#\mathsf{a} + 1$. These imply $\#\mathsf{a} = \#\mathsf{a} + 2$, which has no solutions. Hence there is no level numbering w.r.t. M_2.* □

Proposition 3 *Let M be a supported model of P. If there is a level numbering $\#$ w.r.t. M, then $\#$ is unique.*

The key observation is that the existence of a level numbering is inherently connected to stability. To determine level numberings in practice, we resort to 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 a positive program P and any set of atoms $A \subseteq \mathrm{At}(P)$[2]. 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 $\mathrm{T}_P \uparrow \omega = \bigcup_{i<\omega} \mathrm{T}_P \uparrow i$. Then we have $\mathrm{LM}(P) = \mathrm{T}_P \uparrow \omega = \mathrm{lfp}(\mathrm{T}_P)$ for a positive program P. In case of a finite program P, $\mathrm{lfp}(\mathrm{T}_P)$ is always reached with a finite number of steps. We are now ready to define the *level number* $\mathrm{lev}(\mathsf{a})$, i.e. the least natural number i such that $\mathsf{a} \in \mathrm{T}_P \uparrow i$, for each true atom $\mathsf{a} \in \mathrm{LM}(P)$. This definition extends for rules $r \in \mathrm{SR}(P, \mathrm{LM}(P))$ in analogy to (3): $\mathrm{lev}(r) = \max\{\mathrm{lev}(\mathsf{b}) \mid \mathsf{b} \in \mathrm{B}^+(r)\} + 1$.

Assigning level numbers in this way is compatible with Definition 1 which implies a characterization of stable models based on the existence of level numberings.

Theorem 4 *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 atoms $\mathsf{a} \in M$, let $\#\mathsf{a} = \mathrm{lev}(\mathsf{a})$.*
   (b) *For rules $r \in \mathrm{SR}(P, M)$, let $\#r = \mathrm{lev}(r^+)$.*
2. *If M is a supported model of P and there is a level numbering $\# : M \cup \mathrm{SR}(P, M) \to \mathbb{N}$ w.r.t. M, then $\#$ is unique and M is a stable model of P.*

4 NEW CLAUSAL REPRESENTATION

In this section, we develop a new way to translate a normal logic program into a set of clauses so that a tight correspondence of models is obtained. More precisely, we aim at *faithfulness* in the sense proposed in [10, 11]: 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 $\mathrm{At}(P)$. This is to properly preserve the semantics of the program, including the number of models[3]. As a further requirement, we try to keep the length of the translation $||\mathrm{Tr}(P)||$ as low as possible; preferably *sub-quadratic*, i.e. of order $||P|| \times \log_2 |\mathrm{At}(P)|$.

The translation of a normal program P takes place in two subsequent steps. First, we remove positive body atoms from all rules of P. The result is an atomic normal program $\mathrm{Tr}_{\mathrm{AT}}(P)$ which is then easy to convert into a set of clauses using Clark's completion. However, the first step is much more complicated. 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 consider another example on a level numbering in order to better understand the range taken by level numbers.

Example 5 *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 $\#r_2 = 3$.* □

[2] Note that an interpretation $M \subseteq \mathrm{At}(P)$ is a supported model of $P \iff M = \mathrm{T}_{P^M}(M)$.

[3] This is essential, as models correspond to solutions in ASP.

4.1 Representing Level Numbers

It is natural to use a binary encoding when representing individual level numbers determined by a level numbering $\#$ in terms of propositional atoms. Unfortunately, every atom in $\mathrm{At}(P)$ may be assigned a different level number in the worst case, as demonstrated in Example 5. Thus the level numbers of atoms may vary from 1 to $|\mathrm{At}(P)|$ and 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 by definition, we leave room for 0 which is to act as the least binary value. Therefore, the maximum number of bits in level numbers is

$$\nabla P = \lceil \log_2(|\mathrm{At}(P)| + 2) \rceil. \tag{4}$$

Given the number of bits b, a natural number $0 \leq n < 2^b$, and $0 < i \leq b$, we write $n[i]$ for the i^{th} bit in the binary representation of n in the decreasing order of significance. Our idea is to encode the level number $\#\mathsf{a}$ of an atom $\mathsf{a} \in \mathrm{At}(P)$ using a *vector* $\mathsf{a}_1, \ldots, \mathsf{a}_j$ of new atoms. Such a vector can be understood as a representation of a *binary counter* of j bits, which is to hold $\#\mathsf{a}$ as its binary value. Since atoms may take only two values under the stable model semantics, we aim at the following relationship: $\#\mathsf{a}[i] = 1$ (resp. $\#\mathsf{a}[i] = 0$) iff a_i evaluates to true (resp. false) under stable model semantics.

In order to capture level numberings with binary counters, we need certain primitive operations to be used as subprograms of the forthcoming translation $\mathrm{Tr_{AT}}(P)$. The first set of subprograms, as listed in upper half of Table 1, concentrates on setting counters to particular values. Each subprogram is to be activated only when an additional controlling atom c cannot be inferred by other rules. The first subprogram $\mathrm{SEL}_j(\mathsf{a}, \mathsf{c})$ selects a value $0 \leq n < 2^j$ for the binary counter $\mathsf{a}_1, \ldots, \mathsf{a}_j$ associated with an atom a. Note that the new atoms $\overline{\mathsf{a}_1}, \ldots, \overline{\mathsf{a}_j}$ act as complements of $\mathsf{a}_1, \ldots, \mathsf{a}_j$ and we need them to keep subprograms and the overall translation $\mathrm{Tr_{AT}}(P)$ atomic. The second subprogram $\mathrm{NXT}_j(\mathsf{a}, \mathsf{b}, \mathsf{c})$ binds the values of the binary counters associated with atoms a and b, respectively, so that the latter is the former increased by one (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.

There is also a need to compare values. The fourth 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 their complements 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. The fifth subprogram $\mathrm{EQ}_j(\mathsf{a}, \mathsf{b}, \mathsf{c})$ checks if the counters associated with a and b hold the same value. Only two new atoms $\mathsf{eq}(\mathsf{a}, \mathsf{b})$ and $\overline{\mathsf{eq}(\mathsf{a}, \mathsf{b})}$ are needed for the result.

Table 1. Encoding primitive operations for binary counters

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

4.2 Removing Positive Body Atoms

Any *non-binary* rule r with $|\mathrm{B}^+(r)| > 2$ can be transformed into a set of binary rules in a faithful way using new atoms [10]. Thus we assume without loss of generality that programs under consideration are free of such rules. Moreover, we partition each program P into its *strongly connected components* (SCCs) $C_1, \ldots, C_n$ using positive dependencies (cf. [14]). In the sequel, we describe the translation $\mathrm{Tr_{AT}}(C_i)$ for a SCC C_i. Note that the atoms in the set $\mathrm{H}(C_i) = \{\mathrm{H}(r) \mid r \in C_i\}$ are mutually reachable in the *positive dependency graph* $\mathrm{DG}^+(P)$ having $\mathrm{H}(P)$ and $\{\langle \mathrm{H}(r), \mathsf{b} \rangle \mid r \in P \text{ and } \mathsf{b} \in \mathrm{B}^+(r)\}$ as the sets of vertices and edges, respectively. The translation $\mathrm{Tr_{AT}}(P)$ is then obtained as $\bigcup_i \mathrm{Tr_{AT}}(C_i)$. In the sequel, we describe the contribution of an atom $\mathsf{a} \in \mathrm{H}(C_i)$ to $\mathrm{Tr_{AT}}(C_i)$. If a appears positively in P, we introduce a new atom $\overline{\mathsf{a}}$, i.e. the complement of a, and its definition $\overline{\mathsf{a}} \leftarrow {\sim}\mathsf{a}$ in $\mathrm{Tr_{AT}}(C_i)$.

If $|\mathrm{H}(C_i)| = 1$, i.e. $\mathrm{H}(C_i) = \{\mathsf{a}\}$, it is sufficient to include

$$\mathsf{a} \leftarrow {\sim}\overline{\mathsf{b}}_1, \ldots, {\sim}\overline{\mathsf{b}}_n, {\sim}\mathsf{c}_1, \ldots, {\sim}\mathsf{c}_m \tag{5}$$

as the translation of a rule $r \in C_i$ with $\mathrm{H}(r_i) = \mathsf{a}$. If $\mathsf{a} \in \mathrm{B}^+(r)$, then (5) can be omitted. On the other hand, if $|\mathrm{H}(C_i)| > 1$, we have to introduce two binary counters for a, one for holding $\#\mathsf{a}$ and the other for holding $\#\mathsf{a}+1 \bmod 2^{\nabla C_i}$ where $\nabla C_i = \lceil \log_2(|\mathrm{H}(C_i)| + 2) \rceil$ is defined according to (4). These counters are represented using vectors of new atoms $\mathsf{ctr}(\mathsf{a})_1, \ldots, \mathsf{ctr}(\mathsf{a})_{\nabla C_i}$ and $\mathsf{nxt}(\mathsf{a})_1, \ldots, \mathsf{nxt}(\mathsf{a})_{\nabla C_i}$, respectively, including their complements. To set the values of these counters appropriately, we include subprograms $\mathrm{SEL}_{\nabla C_i}(\mathsf{ctr}(\mathsf{a}), \overline{\mathsf{a}})$ and $\mathrm{NXT}_{\nabla C_i}(\mathsf{ctr}(\mathsf{a}), \mathsf{nxt}(\mathsf{a}), \overline{\mathsf{a}})$ in $\mathrm{Tr_{AT}}(C_i)$. Note that these subprograms are activated only if $\overline{\mathsf{a}}$ cannot be inferred, i.e. a is inferable. The translation of a rule $r \in C_i$ with $\mathrm{H}(r) = \mathsf{a}$ is more involved now. We introduce a new atom $\mathsf{bt}(r)$ denoting that the body of r is satisfied and its complement $\overline{\mathsf{bt}(r)}$. The definition of $\mathsf{bt}(r)$ obtained from (5) by substituting $\mathsf{bt}(r)$ for the head a. In addition to such a rule, we need $\overline{\mathsf{bt}(r)} \leftarrow {\sim}\mathsf{bt}(r)$ and $\mathsf{a} \leftarrow {\sim}\overline{\mathsf{bt}(r)}$ in $\mathrm{Tr_{AT}}(C_i)$ to capture support for a (recall the definition of a supported model). Moreover, we need certain *constraints* to ensure that the values held by counters constitute a valid level numbering. The minimality of $\#\mathsf{a}$ in (2) is checked indirectly using a new atom $\mathsf{min}(\mathsf{a})$ and a constraint $\mathsf{x} \leftarrow {\sim}\mathsf{x}, {\sim}\overline{\mathsf{a}}, {\sim}\mathsf{min}(\mathsf{a})$ activating when a is true.

The constraints associated with a rule r having $\mathrm{H}(r) = \mathsf{a}$ will be conditioned with a negative body atom $\overline{\mathsf{bt}(r)}$ which — whenever not inferable — captures the fact that

$r \in \mathrm{SR}(P, M)$. If r is *atomic*, we use the subprogram $\mathrm{FIX}_{\nabla C_i}(\mathsf{ctr}(\mathsf{a}), 1, \overline{\mathsf{bt}(r)})$ to ensure $\#\mathsf{a} = 1$ whenever $r \in \mathrm{SR}(P, M)$; and a rule $\mathsf{min}(\mathsf{a}) \leftarrow \sim\overline{\mathsf{bt}(r)}$ to infer the minimality of the value held by $\mathsf{ctr}(\mathsf{a})$, as insisted by (2).

On the other hand, if r is *unary* with $\mathrm{H}(r) = \mathsf{a}$ and $\mathrm{B}^+(r) = \{\mathsf{b}\}$, two cases arise. If $\mathsf{b} \notin \mathrm{H}(C_i)$, then r is translated like unary rules. Otherwise, we have $\mathsf{b} \in \mathrm{H}(C_i)$ and the value of (3) is held by $\mathsf{nxt}(\mathsf{b})$. This is needed to express the contribution of r in (2) w.r.t. $\mathrm{H}(r) = \mathsf{a}$. For this purpose, we have to compare the values held by the counters $\mathsf{nxt}(\mathsf{b})$ and $\mathsf{ctr}(\mathsf{a})$ using subprograms $\mathrm{LT}_{\nabla C_i}(\mathsf{ctr}(\mathsf{nxt}(\mathsf{b})), \mathsf{ctr}(\mathsf{a}), \overline{\mathsf{bt}(r)})$ and $\mathrm{EQ}_{\nabla C_i}(\mathsf{ctr}(\mathsf{nxt}(\mathsf{b})), \mathsf{ctr}(\mathsf{a}), \overline{\mathsf{bt}(r)})$. Then the requirement that $\#\mathsf{a} \geq \#\mathsf{b} + 1$, as insisted by (2), can be expressed in terms of a constraint $\mathsf{x} \leftarrow \sim\mathsf{x}, \sim\overline{\mathsf{bt}(r)}, \sim\overline{\mathsf{lt}(\mathsf{nxt}(\mathsf{b}), \mathsf{ctr}(\mathsf{a}))_1}$ where x is a new atom. The other half of (2) is taken care by a rule of the form $\mathsf{min}(\mathsf{a}) \leftarrow \sim\overline{\mathsf{bt}(r)}, \sim\overline{\mathsf{eq}(\mathsf{nxt}(\mathsf{b}), \mathsf{ctr}(\mathsf{a}))}$.

The case of a binary rule r with $\mathrm{H}(r) = \mathsf{a}$ and $\mathrm{B}^+(r) = \{\mathsf{b}_1, \mathsf{b}_2\}$ follows. We may assume that $\mathrm{B}^+(r) \subseteq \mathrm{H}(C_i)$, because otherwise r can be handled like an atomic or a unary rule. In this case, we need to compare the values of $\mathsf{ctr}(\mathsf{b}_1)$ and $\mathsf{ctr}(\mathsf{b}_2)$ using $\mathrm{LT}_{\nabla C_i}(\mathsf{ctr}(\mathsf{b}_1), \mathsf{ctr}(\mathsf{b}_2), \overline{\mathsf{bt}(r)})$ to decide which one is relevant for (3). Then the constraints associated with r as well as the rules for $\mathsf{min}(\mathsf{a})$ are the same as for the two unary rules $\mathsf{a} \leftarrow \mathsf{b}_1, \sim\mathsf{lt}(\mathsf{ctr}(\mathsf{b}_1), \mathsf{ctr}(\mathsf{b}_2)), \sim\mathrm{B}^-(r)$ and $\mathsf{a} \leftarrow \mathsf{b}_2, \sim\overline{\mathsf{lt}(\mathsf{ctr}(\mathsf{b}_1), \mathsf{ctr}(\mathsf{b}_2))}, \sim\mathrm{B}^-(r)$. No other rules are needed.

Given a (binary) normal logic program P and its translation $\mathrm{Tr}_{\mathrm{AT}}(P)$ as explained above, the stable models in $\mathrm{SM}(P)$ and $\mathrm{SM}(\mathrm{Tr}_{\mathrm{AT}}(P))$ are in a bijective relationship and 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 of order $||P|| \times \log_2 |\mathrm{At}(P)|$. On the other hand, a faithful and *modular* translation function from normal programs into atomic ones is impossible [10, 11]. Thus $\mathrm{Tr}_{\mathrm{AT}}$ is necessarily non-modular and $\mathrm{Tr}_{\mathrm{AT}}(P)$ cannot be formed on a rule-by-rule basis. One source of non-modularity is hidden in the numbers of bits: 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 to form $\mathrm{Tr}_{\mathrm{AT}}(P \cup Q)$.

Example 6 *Let us reconsider the program P from Example 2. Both rules and atoms are involved in the only SCC of P, say C, so that $\nabla C = 2$. The translation $\mathrm{Tr}_{\mathrm{AT}}(P) = \mathrm{Tr}_{\mathrm{AT}}(C)$ contains the following rules associated with* a*:* $\overline{\mathsf{a}} \leftarrow \sim\mathsf{a}$; $\mathsf{x} \leftarrow \sim\mathsf{x}, \sim\overline{\mathsf{a}}, \sim\mathsf{min}(\mathsf{a})$; $\mathsf{bt}(r_2) \leftarrow \sim\overline{\mathsf{a}}$; $\overline{\mathsf{bt}(r_2)} \leftarrow \sim\mathsf{bt}(r_2)$; $\mathsf{b} \leftarrow \sim\overline{\mathsf{bt}(r_2)}$; $\mathsf{x} \leftarrow \sim\mathsf{x}, \sim\overline{\mathsf{bt}(r_2)}, \sim\overline{\mathsf{lt}(\mathsf{nxt}(\mathsf{a}), \mathsf{ctr}(\mathsf{b}))_1}$; *and* $\mathsf{min}(\mathsf{b}) \leftarrow \sim\overline{\mathsf{bt}(r_2)}, \sim\overline{\mathsf{eq}(\mathsf{nxt}(\mathsf{a}), \mathsf{ctr}(\mathsf{b}))}$ *in addition to four subprograms for choosing the values of* $\mathsf{ctr}(\mathsf{a})$ *and* $\mathsf{nxt}(\mathsf{a})$ *as well as comparing the latter with* $\mathsf{ctr}(\mathsf{b})$*. The rules for* b *are symmetric, just exchange the roles of* a *and* b*; and r_2 and r_1. The only stable model $M = \emptyset$ of P is then captured as the only stable model* $N = \{\overline{\mathsf{a}}, \overline{\mathsf{b}}, \overline{\mathsf{bt}(r_1)}, \overline{\mathsf{bt}(r_2)}\}$ *of* $\mathrm{Tr}_{\mathrm{AT}}(P)$.

4.3 From Atomic Rules to Clauses

Atomic normal programs provide a promising intermediary representation that is 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 in a faithful way. However, new atoms must be introduced in order to keep the translation linear. Since this is a quite standard procedure, we skip the details.

5 EXPERIMENTS

We have implemented the translation described in Section 4. The implementation consists of two translators 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 the SMODELS system [21]; typically produced by the front-end LPARSE. The latter translator, LP2SAT takes the output of LP2ATOMIC as its input and produces Clark's completion for the program. The output is in the *DIMACS format* which is understood by most SAT solvers.

As a test program, we use a normal logic program given in Figure 1. The program is given in the input syntax of LPARSE and it formalizes the problem of finding any subgraph of D_n, i.e. the complete directed graph with n vertices and n^2 edges, in which all vertices are still reachable from each other. In our benchmark, the task is to compute all stable models of the program instantiated by LPARSE when n varies from 1 to 5. As a result, the number of SCCs and *positive loops* increases.

Figure 1. Normal logic program used in the benchmark

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

We run five benchmark instances generated by LPARSE on six different systems also listed in Table 2: SMODELS, CMODELS [13], and four combinations of LP2ATOMIC and LP2SAT with other solvers. The first combines plain LP2ATOMIC with SMODELS just to get an idea how much overhead results from the removal of positive body atoms. The second combination uses both translators and a state-of-the-art SAT solver CHAFF [19] for the actual computation of classical models corresponding to stable models. The third is the same as the second except another solver, namely RELSAT [3], is used as the back-end. The last system incorporates a strengthened well-founded reduction to this setting: we use SMODELS to simplify the intermediate program representations before and after invoking LP2ATOMIC. This has a favorable effect on the number of clauses generated, as notable from Table 2.

Our benchmark is easy for SMODELS which reaches a performance of 47 kMPS (models per second) on a 1.67 GHz CPU. However, the main objective here is to compare CMODELS with our approach, as it is also based on the idea of using SAT solvers to compute stable models instead of a special purpose engine like SMODELS. We did not include ASSAT [15], as it can compute only one stable model for a program given as input. When $n = 4$ the performance of the systems based on LP2ATOMIC and LP2SAT is between 1.0–2.8 kMPS, which clearly exceeds that of CMODELS, i.e. only 5.5 MPS. When $n = 5$, CMODELS exceeds the time limit of 24 hours and CHAFF

Table 2. Timings in seconds when computing all stable models

Vertices	1	2	3	4	5
SMODELS	0.004	0.003	0.003	0.033	12
CMODELS	0.031	0.030	0.124	293	-
LP2ATOMIC+SMODELS	0.004	0.008	0.013	0.393	353
LP2SAT+CHAFF	0.011	0.009	0.023	1.670	-
LP2SAT+RELSAT	0.004	0.005	0.018	0.657	1879
WF+LP2SAT+RELSAT	0.009	0.013	0.018	0.562	1598
Models	1	1	18	1606	565080
SCCs with $\lvert\mathrm{H}(C)\rvert > 1$	0	0	3	4	5
Rules (LPARSE)	3	14	39	84	155
Rules (LP2ATOMIC)	3	18	240	664	1920
Clauses (LP2SAT)	4	36	818	2386	7642
Clauses (WF+LP2SAT)	2	10	553	1677	5971

runs out of memory (1 GB) as the back-end of the fourth system. 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 2; and 0.020 seconds for ASSAT [15].

6 DISCUSSION

In this paper, we tackle a very challenging problem of translating normal logic programs into sets of clauses. As a solution, we propose a novel translation that has a unique combination of properties. First, a bijective correspondence of models is obtained in contrast to the approach Ben-Eliyahu and Dechter [4] whose translation $\mathrm{Tr}_{\mathrm{BD}}(P)$ may posses several classical models corresponding to one stable model $M \in \mathrm{SM}(P)$. Moreover, $\mathrm{At}(P)$ is not preserved, as $\mathrm{At}(P) \cap \mathrm{At}(\mathrm{Tr}_{\mathrm{BD}}(P)) = \emptyset$. Second, our translation is sub-quadratic, which differentiates it from existing translations of quadratic [4, 14] and even exponential [5] worst-case complexities. Third, $\mathrm{Tr}_{\mathrm{AT}}(P)$ can be computed at once in spite of non-modularity. This differs from approaches [15, 13] where *loop formulas* are gradually added to the completion of the program and an exponential blow-up may result even if one stable model is computed.

The new 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. A particular novelty of a level numbering conforming to Definition 1 is that the values assigned to atoms are uniquely determined. This is in sharp contrast with earlier characterizations of stable models [4, 2, 7], where similar numberings are used to distinguish stable models, but the value assignment can be done in several ways. Canonical level numberings are crucial for the objective of obtaining a bijective correspondence between models.

Despite promising properties and experimental results, the translation function $\mathrm{Tr}_{\mathrm{AT}}$ is not yet optimal. In the future, we intend to study techniques to reduce the number of binary counters and the number of bits when translating a SCC.

ACKNOWLEDGMENTS

The research reported in this paper is partially funded by the Academy of Finland (project #53695) and the European commission (contract IST-FET-2001-37004).

REFERENCES

[1] K.R. Apt, H.A. Blair, and A. Walker, 'Towards a theory of declarative knowledge', in *Foundations of Deductive Databases and Logic Programming*, ed., J. Minker, 89–148, Morgan Kaufmann, 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.J. Bayardo and R. Schrag, 'Using CSP look-back techniques to solve real-world sat instances', in *Proceedings of the 12th National Conference*, pp. 203–208. AAAI, (1997).

[4] R. Ben-Eliyahu and R. Dechter, 'Propositional semantics for disjunctive logic programs', *Annals of Mathematics and Artificial Intelligence*, **12**(1–2), 53–87, (1994).

[5] S. Brass and J. Dix, 'Semantics of (disjunctive) logic programs based on partial evaluation', *Journal of Logic Programming*, **38**(3), 167–213, (1999).

[6] Y. Dimopoulos, B. Nebel, and J. Koehler, 'Encoding planning problems in non-monotonic logic programs', in *Proceedings of the 4th European Conference on Planning*, pp. 169–181, Toulouse, France, (September 1997). Springer.

[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*, pp. 1070–1080, Seattle, USA, (August 1988). MIT Press.

[10] T. Janhunen, 'Comparing the expressive powers of some syntactically restricted classes of logic programs', in *Computational Logic, First International Conference*, eds., J. Lloyd et al., pp. 852–866, London, UK, (July 2000). Springer.

[11] T. Janhunen, 'Translatability and intranslatability results for certain classes of logic programs', Series A: Research report 82, Helsinki University of Technology, Laboratory for Theoretical Computer Science, Espoo, Finland, (2003).

[12] 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).

[13] Y. Lierler and M. Maratea, 'Cmodels-2: Sat-based answer set solver enhanced to non-tight programs', in *Proceedings of LPNMR-7*, pp. 346–350, Fort Lauderdale, Florida, (January 2004). Springer. LNAI 2923.

[14] F. Lin and J. Zhao, 'On tight logic programs and yet another translation from normal logic programs to propositional logic', in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pp. 853–858, (2003).

[15] F. Lin and Y. Zhao, 'ASSAT: Computing answer sets of a logic program by sat solvers', in *Proceedings of the 18th National Conference on Artificial Intelligence*, pp. 112–117, Edmonton, Alberta, Canada, (July/August 2002). AAAI.

[16] J.W. Lloyd, *Foundations of Logic Programming*, Springer, Berlin, 1987.

[17] 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).

[18] W. Marek and M. Truszczyński, 'Stable models and an alternative logic programming paradigm', in *The Logic Programming Paradigm: a 25-Year Perspective*, Springer, (1999).

[19] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, 'Chaff: Engineering an efficient sat solver', in *Proceedings of the 39th Design Automation Conference*, Las Vegas, (2001).

[20] 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).

[21] P. Simons, I. Niemelä, and T. Soininen, 'Extending and implementing the stable model semantics', *Artificial Intelligence*, **138**(1–2), 181–234, (2002).