An Approach to Capture Stable Models with Classical Ones

Tomi Janhunen

Helsinki University of Technology

Department of Computer Science and Engineering

Laboratory for Theoretical Computer Science

Dagstuhl Seminar on Nonmonotonic Reasoning, Answer Set Programming and Constraints

September 16, 2002

BACKGROUND AND MOTIVATION

➢ Normal logic programs (under stable model semantics) vs. sets of clauses (under classical models).

➢ If P is *positive-order-consistent* (or *tight*), then the stable models of P and the classical models of $\mathrm{Comp}(P)$ coincide.

Y. Babovich, E. Erdem, and V. Lifschitz [NMR Workshop, 2000]: *Fages' Theorem and Answer Set Programming*.

➢ Removing positive literals from the rule bodies is not easy.

T. Janhunen [CL 2000]: *Comparing the Expressive Powers of Some Syntactically Restricted Classes of Logic Programs*.

➢ *Partial evaluation* takes an exponential space in the worst case.

OUTLINE OF THE TALK

1. Classes of Logic Programs
2. Properties of Translation Functions
3. Classification Method
4. Non-modular Translation
5. Optimizations in Sight
6. Implementation and Experiments
7. Related Work
8. Conclusions

1. CLASSES OF LOGIC PROGRAMS

Definition: *A propositional logic program P is a set of expressions (such as rules, clauses or sentences) built of propositional atoms.*

➢ We let $\mathrm{Hb}(P)$ stand for the Herbrand base of a logic program P.

➢ General requirements for a class $\mathcal{C}$ of logic programs:

1. Closure under unions: if $P \in \mathcal{C}$ and $Q \in \mathcal{C}$, then $P \cup Q \in \mathcal{C}$.
2. There is a semantical operator $\mathrm{Sem}_{\mathcal{C}}$ that maps any $P \in \mathcal{C}$ to $\mathrm{Sem}_{\mathcal{C}}(P) \subseteq \mathbf{2}^{\mathrm{Hb}(P)}$, i.e., the **set of models** of P.

$\Longrightarrow$ The class of sets of propositional clauses under classical models ($\mathcal{SAT}$) can be viewed as a class of logic programs.

Some Syntactic Subclasses of Normal Programs

- The representatives of the class of normal logic programs $\mathcal{N}$ that consist of rules of the form $h \leftarrow b_1, \ldots, b_n, {\sim}c_1, \ldots, {\sim}c_m$.
- The number of positive body literals n is constrained as follows:
  1. The class of **atomic programs** $\mathcal{A}$ ($n = 0$ for every rule).
  2. The class of **unary programs** $\mathcal{U}$ ($n \leq 1$ for every rule).
  3. The class of **binary programs** $\mathcal{B}$ ($n \leq 2$ for every rule).

  ☞ $\mathcal{A} \subset \mathcal{U} \subset \mathcal{B} \subset \mathcal{N}$.
- **Strictly unary** ($n = 1$), **strictly binary** ($n = 2$) and **non-binary** $n > 2$ rules are also distinguished.
- Our criteria for classes of logic programs are met (the semantics of the programs is determined by stable models in each class).

2. PROPERTIES OF TRANSLATION FUNCTIONS

Definition: *A translation function* $\mathrm{Tr} : \mathcal{C} \to \mathcal{C}'$ *is* **PFM** $\iff$ Tr *is*

- **polynomial (P)**,

  The time required to compute $\mathrm{Tr}(P)$ *is polynomial in* $||P||$.
- **faithful (F)**, *and*
  1. *For all programs* $P \in \mathcal{C}$, $\mathrm{Hb}(P) \subseteq \mathrm{Hb}(\mathrm{Tr}(P))$.
  2. *The models/interpretations in* $\mathrm{Sem}_{\mathcal{C}}(P)$ *and* $\mathrm{Sem}_{\mathcal{C}'}(\mathrm{Tr}(P))$ *are in a bijective relationship and coincide up to* $\mathrm{Hb}(P)$.
- **modular (M)**.
  1. *For every* $P \in \mathcal{C}$ *and* $Q \in \mathcal{C}$, $\mathrm{Tr}(P \cup Q) = \mathrm{Tr}(P) \cup \mathrm{Tr}(Q)$.
  2. *If* $\mathcal{C}' \subset \mathcal{C}$, *then* $\mathrm{Tr}(P') = P'$ *for every* $P' \in \mathcal{C}'$.

3. CLASSIFICATION METHOD

Given two classes $\mathcal{C}$ and $\mathcal{C}'$ of programs, the goal is to establish either

- $\mathcal{C} \underset{\text{PFM}}{\longrightarrow} \mathcal{C}'$ (there is a PFM translation function $\mathrm{Tr} : \mathcal{C} \to \mathcal{C}'$), **or**
- $\mathcal{C} \underset{\text{PFM}}{\not\longrightarrow} \mathcal{C}'$ (such a translation function does not exist).

These relations induce further relations for classes of logic programs:

Relation	Definition	Explanation
$\mathcal{C} \underset{\text{PFM}}{\Longrightarrow} \mathcal{C}'$	$\mathcal{C} \underset{\text{PFM}}{\longrightarrow} \mathcal{C}'$ and $\mathcal{C}' \underset{\text{PFM}}{\not\longrightarrow} \mathcal{C}$	*less expressive*
$\mathcal{C} \underset{\text{PFM}}{\longleftrightarrow} \mathcal{C}'$	$\mathcal{C} \underset{\text{PFM}}{\longrightarrow} \mathcal{C}'$ and $\mathcal{C}' \underset{\text{PFM}}{\longrightarrow} \mathcal{C}$	*equally expressive*
$\mathcal{C} \underset{\text{PFM}}{\not\longleftrightarrow} \mathcal{C}'$	$\mathcal{C} \underset{\text{PFM}}{\not\longrightarrow} \mathcal{C}'$ and $\mathcal{C}' \underset{\text{PFM}}{\not\longrightarrow} \mathcal{C}$	*mutually incomparable*

☞ Classes can be ordered on the basis of their expressive power.

Expressive Power Hierarchy

- It is straightforward to encode propositional satisfiability problems as an (atomic) normal programs: $\mathcal{SAT} \underset{\text{PFM}}{\longrightarrow} \mathcal{A}$.
- $\mathcal{A} \subset \mathcal{U} \subset \mathcal{B} \subset \mathcal{N} \implies \mathcal{A} \underset{\text{PFM}}{\longrightarrow} \mathcal{U} \underset{\text{PFM}}{\longrightarrow} \mathcal{B} \underset{\text{PFM}}{\longrightarrow} \mathcal{N}$ (by identity).
- Non-binary rules can be translated away: $\mathcal{N} \underset{\text{PFM}}{\longrightarrow} \mathcal{B}$.
- Binary and unary rules cannot be translated away in a **faithful and modular** way: $\mathcal{B} \underset{\text{FM}}{\not\longrightarrow} \mathcal{U}$ and $\mathcal{U} \underset{\text{FM}}{\not\longrightarrow} \mathcal{A}$.
- Due to non-monotonicity $\mathcal{A} \underset{\text{FM}}{\not\longrightarrow} \mathcal{SAT}$.
- The resulting hierarchy of the five classes under consideration:

$$\mathcal{SAT} \underset{\text{PFM}}{\Longrightarrow} \mathcal{A} \underset{\text{PFM}}{\Longrightarrow} \mathcal{U} \underset{\text{PFM}}{\Longrightarrow} \mathcal{B} \underset{\text{PFM}}{\longleftrightarrow} \mathcal{N}$$

4. NON-MODULAR TRANSLATION

A normal logic program P is translated into an **atomic** one $\mathrm{Tr}(P)$.

➢ The goal is a polynomial and faithful translation such that $||\mathrm{Tr}(P)||$ is of order $||P|| \times \log_2 |\mathrm{Hb}(P)|$.

➢ As a characteristic of a stable model M of P, the numbers

$$\mathrm{level}(a) = \begin{cases} \min\limits_{\substack{\mathrm{head}(r)=a \\ M \models \mathrm{body}(r)}} \mathrm{level}(r), & \text{if } M \models a. \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathrm{level}(r) = \begin{cases} \max\limits_{b \in \mathrm{body}^+(r)} \mathrm{level}(b) + 1, & \text{if } \mathrm{body}^+(r) \neq \emptyset. \\ 0 & \text{otherwise.} \end{cases}$$

are uniquely defined for any $a \in \mathrm{Hb}(P)$ and $r \in P$.

➢ Level numbers lead to a characterization of stable models:

Theorem:

1. If M is a *stable model* of P, then M is a *supported model* of P and there is a level numbering (as described above) for atoms $a \in \mathrm{Hb}(P)$ and rules $r \in P$.
2. If M is a *supported model* of P and there is a level numbering (as described above) for atoms $a \in \mathrm{Hb}(P)$ and rules $r \in P$, then M is *stable model* of P.

➢ This suggests a translation that consists of two independent parts:

1. The first part captures supported models of P.
2. The second part checks if one can assign a level numbering (as described above) for atoms $a \in \mathrm{Hb}(P)$ and $r \in P$.

Translation of Atoms

An atom $a \in \mathrm{Hb}(P)$ is translated as follows:

➢ The truth value of a is chosen:

$$a \leftarrow \sim\overline{a}. \quad \overline{a} \leftarrow \sim a.$$

➢ Let $N = \lceil \log_2(|\mathrm{Hb}(P)| + 1) \rceil$ and let us introduce a counter (a vector of atoms) $a_{\mathrm{cnt}} = a_1 \ldots a_N$ and $\overline{a_{\mathrm{cnt}}} = \overline{a_1} \ldots \overline{a_N}$.

➢ The value of a_{cnt} is chosen: for each $0 < i \leq N$,

$$a_i \leftarrow \sim\overline{a_i}. \quad \overline{a_i} \leftarrow \sim a_i.$$

➢ Similarly, we need to define another counter a_{succ} that takes the value of a_{cnt} incremented by one: subprogram $\mathrm{SUCC}(a_{\mathrm{cnt}}, a_{\mathrm{succ}})$.

Translation of Rules

A new atom r (and the complement $\overline{r}$) is introduced for each rule $a \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m$ in P.

Rules that are needed to capture *supported models* of P:

➢ Evaluate the body of the rule:

$$r \leftarrow \sim\overline{b}_1, \ldots, \sim\overline{b}_n, \sim c_1, \ldots, \sim c_m. \quad \overline{r} \leftarrow \sim r.$$

➢ Every rule must be satisfied:

$$\bot \leftarrow \sim a, \sim\overline{r}.$$

plus $\top \leftarrow \sim\bot$ and $f \leftarrow \sim f, \sim\top$ where $\top$, $\bot$ and f are new.

➢ Every atom must have a supporting rule:

$$a_{\mathrm{supp}} \leftarrow \sim\overline{r}. \quad \bot \leftarrow \sim\overline{a}, \sim a_{\mathrm{supp}}.$$

Translation of Rules (Continued)

- A counter is introduced for each rule:

  $$r_{\mathrm{cnt}} = r_1 \ldots r_N \text{ and } \overline{r_{\mathrm{cnt}}} = \overline{r_1} \ldots \overline{r_N}.$$

  Thus we need $r_i \leftarrow \sim\overline{r_i}$ and $\overline{r_i} \leftarrow \sim r_i$ for each $0 < i \leq N$.
- The value of r_{cnt} is supposed to be $\mathrm{level}(r)$ in binary.
- If $n > 0$, we need for each $b \in \{b_1, \ldots, b_n\}$ subprograms $\mathrm{LTHAN}(r_{\mathrm{cnt}}, b_{\mathrm{succ}})$ and $\mathrm{EQ}(r_{\mathrm{cnt}}, b_{\mathrm{succ}})$ in addition to

  $$\perp \leftarrow \sim\overline{\mathrm{lthan}(r_{\mathrm{cnt}}, b_{\mathrm{succ}})}.$$

  $$r_{\mathrm{eq}} \leftarrow \sim\overline{\mathrm{eq}(r_{\mathrm{cnt}}, b_{\mathrm{succ}})}. \quad \perp \leftarrow \sim r_{\mathrm{eq}}.$$
- If $n = 0$, we need the subprogram $\mathrm{ZERO}(r_{\mathrm{cnt}})$ and the rules

  $$\perp \leftarrow \sim\mathrm{zero}(r_{\mathrm{cnt}}).$$

Translation of Atoms (Continued)

- The value of a_{cnt} is supposed to be $\mathrm{level}(a)$ in binary.
- For each rule r having a as head, we need the subprograms $\mathrm{LTHAN}(r_{\mathrm{cnt}}, a_{\mathrm{cnt}})$ and $\mathrm{EQ}(r_{\mathrm{cnt}}, a_{\mathrm{cnt}})$ plus the following rules:

  $$\perp \leftarrow \sim\overline{r}, \sim\overline{\mathrm{lthan}(r_{\mathrm{cnt}}, a_{\mathrm{cnt}})}.$$

  $$a_{\mathrm{eq}} \leftarrow \sim\overline{r}, \sim\overline{\mathrm{eq}(r_{\mathrm{cnt}}, a_{\mathrm{cnt}})}. \quad \perp \leftarrow \sim\overline{a}, \sim a_{\mathrm{eq}}.$$

  Note that $\sim\overline{a}$ is not needed when $\sim\overline{r}$ is present.
- For every $a \in \mathrm{Hb}(P)$, we need $\mathrm{ZERO}(a_{\mathrm{cnt}})$ and the rules

  $$\perp \leftarrow \sim a, \sim\mathrm{zero}(a_{\mathrm{cnt}}).$$

5. OPTIMIZATIONS IN SIGHT

- For certain graph problems, the preceding technique yields translations of size $30 \times ||P|| \times \log_2 |\mathrm{Hb}(P)|$ approximately.
- Counters can be avoided when simple rules are translated:
  1. Atomic r: $h \leftarrow \sim c_1, \ldots, \sim c_m$: $\perp \leftarrow \sim\overline{r}, \sim\mathrm{zero}(h_{\mathrm{cnt}})$.
  2. Unary r: $h \leftarrow b, \sim c_1, \ldots, \sim c_m$:
     $\perp \leftarrow \sim\overline{r}, \sim\overline{\mathrm{lthan}(h_{\mathrm{cnt}}, b_{\mathrm{succ}})}$. $\quad h_{\mathrm{eq}} \leftarrow \sim\overline{\mathrm{eq}(h_{\mathrm{cnt}}, b_{\mathrm{succ}})}$.
  3. Binary r: $h \leftarrow b, b', \sim c_1, \ldots, \sim c_m$: the rules

     $$h \leftarrow b, \sim c_1, \ldots, \sim c_m, \sim\mathrm{lthan}(b_{\mathrm{succ}}, b'_{\mathrm{succ}}).$$
     $$h \leftarrow b', \sim c_1, \ldots, \sim c_m, \sim\overline{\mathrm{lthan}(b_{\mathrm{succ}}, b'_{\mathrm{succ}})}.$$

     are translated like unary rules above.
- The general scheme applies to non-binary rules (if such exist).

Exploiting Dependency Information

- It is possible to localize the translation to strongly connected components (SCCs) of the program.
- Only **positive dependencies** are taken into account.
- A SCC may contain several **loops** $a_1 \rightsquigarrow a_2 \rightsquigarrow \ldots \rightsquigarrow a_n = a_1$.
  Example: $P = \{\ a \leftarrow \sim b. \ \ b \leftarrow \sim a.\ \}$ is free of loops.
- Any rule $h \leftarrow b_1, \ldots, b_n, \sim c_1, \ldots, \sim c_m$ whose head h is not involved in any loop is easily made atomic:

  $$h \leftarrow \sim\overline{b_1}, \ldots, \sim\overline{b_n}, \sim c_1, \ldots, \sim c_m.$$

  $$\overline{b_1} \leftarrow \sim b_1. \quad \ldots \quad \overline{b_n} \leftarrow \sim b_n.$$
- Rules forming a self-loop ($h \in \{b_1, \ldots, b_n\}$) can/must be deleted!

6. IMPLEMENTATION AND EXPERIMENTS

➢ We have implemented two translator programs:

1. LP2ATOMIC transforms an SMODELS program into atomic one.
2. LP2SAT produces Clark's completion in DIMACS format.

Only basic rules and compute statements are covered!

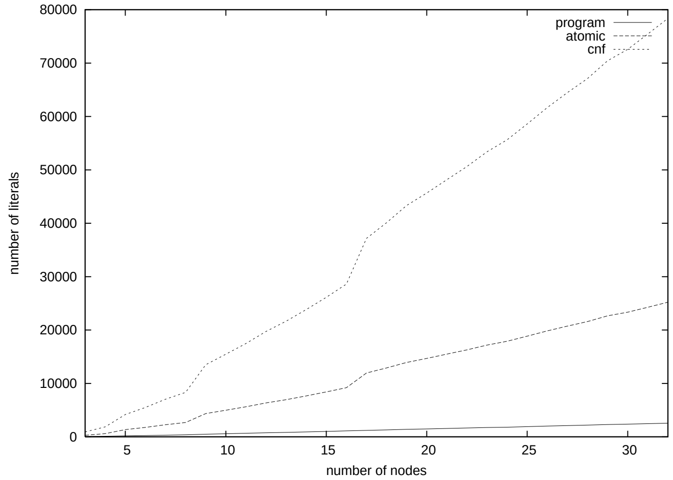
➢ The problem of finding a Hamiltonian circuit for a random planar graph (Niemelä's encoding) is used as a benchmark to compare:

- SMODELS (version 2.26) vs.
- LP2ATOMIC + LP2SAT + ZCAFF (version 2001.2.17).

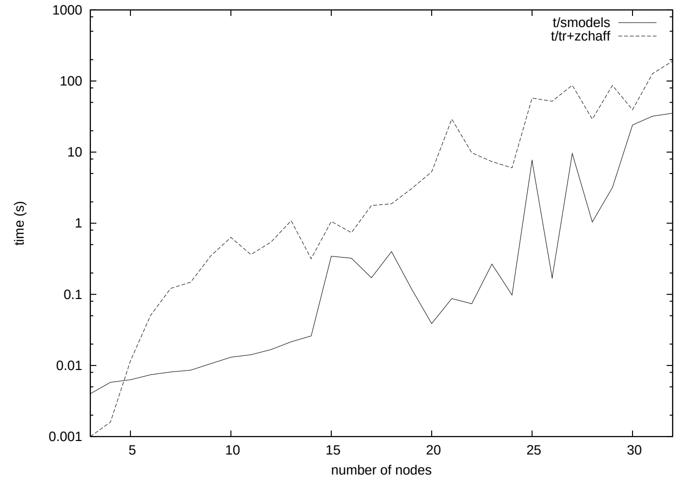
➢ The number of nodes was varied from 3 to 32 (100 instances).

➢ Tests were run on a 451 MHz Pentium III PC running under Linux operating system (kernel version 2.2.19).

Size of the Translation



Time Required for Computing One Model



7. RELATED WORK

➢ R. Ben-Eliyahu and R. Dechter [AMAI, 1994]: *Propositional semantics for disjunctive logic programs.*

- Binary numbers are not used $\Longrightarrow$ at least quadratic encoding.
- The stable models of P and the classical models of $\mathrm{Tr}_{\mathrm{ED}}(P)$ are not in a bijective relationship.

Example: Let $P = \{a \leftarrow b, c.\ \ b \leftarrow d.\ \ c \leftarrow d.\ \ d \leftarrow \sim e.\ \ d \leftarrow a\}$. The atoms b and c in the unique stable model $M = \{a, b, c, d\}$ can be ordered in two different ways (in a **total ordering**).

➢ Y. Babovich, E. Erdem, and V. Lifschitz [NMR Workshop, 2000]: *Fages' Theorem and Answer Set Programming.*

- Programs containing loops are not (necessarily) covered.
- Tightness is based on a different numbering of atoms.

➢ U. Egly, et al. [AAAI, 2001]: *Computing Stable Models with Quantified Boolean Formulas: Some Experimental Results*.

- In this approach, disjunctive stable models are captured with quantified Boolean formulas $\exists p_1 \ldots \exists p_n \forall q_1 \ldots \forall q_m \, \phi$.
- In particular, the minimality requirement of disjunctive stable models is easy to express in terms of such primitives.
- However, the computation of stable models is easier in the case of normal logic programs.

➢ F. Lin and Y. Zhao [AAAI, 2002]: *ASSAT: Computing Answer Sets of a Logic Program by SAT Solvers*.

- The idea is to extend $\mathrm{Comp}(P)$ gradually with *loop formulas*.
- In the worst case, there is an exponential number of loops (Hamiltonian paths for complete graphs).

8. CONCLUSIONS

➢ A PF translation was established: $\mathcal{N} \overset{\rightarrow}{\scriptstyle\mathrm{PF}} \mathcal{A}$, $\mathcal{B} \overset{\rightarrow}{\scriptstyle\mathrm{PF}} \mathcal{A}$, and $\mathcal{U} \overset{\rightarrow}{\scriptstyle\mathrm{PF}} \mathcal{A}$.

➢ Clark's completion is sufficient for $\mathcal{A} \overset{\rightarrow}{\scriptstyle\mathrm{PF}} \mathcal{SAT}$.

➢ **Theorem:** $\mathcal{SAT} \overset{\leftrightarrow}{\scriptstyle\mathrm{PF}} \mathcal{A} \overset{\leftrightarrow}{\scriptstyle\mathrm{PF}} \mathcal{U} \overset{\leftrightarrow}{\scriptstyle\mathrm{PF}} \mathcal{B} \overset{\leftrightarrow}{\scriptstyle\mathrm{PF}} \mathcal{N}$.

➢ Distinctive features of the counter-based approach:

1. bijective relationship of models and
2. $||\mathrm{Tr}(P)||$ is of order $\displaystyle\sum_{S \text{ is a SCC of } P} ||P_S|| \times \log_2(|S| + 1)$.

➢ Transitive closure can be properly captured with classical models.

➢ Experimental results are promising, but further optimizations should be pursued for in order to really compete with SMODELS.