Lecture 12: Translation into Propositional Logic

1. Level numbers and stability
2. Translation into atomic programs
3. Reachability benchmark

Motivation

- The goal is to combine the knowledge representation capabilities of normal programs with the efficiency of SAT solvers.
- To realize this setting, we provide a *faithful* and *polynomial-time* translation $\mathrm{Tr}_{\mathrm{AT}}$ from normal programs into *atomic programs* having rules of the form $a \leftarrow \sim C$ only.
- Such a transformation is inherently *non-modular* but $\mathrm{Tr}_{\mathrm{AT}}(P)$ is always tight so that $P \equiv_{\mathrm{v}} \mathrm{Comp}(\mathrm{Tr}_{\mathrm{AT}}(P))$.
- This leads to an alternative strategy for computing stable models with SAT solvers along with approaches based on loop formulas.

1. LEVEL NUMBERS AND STABILITY

- The *tightness* condition for a normal program P and a supported model $M \models \mathrm{Comp}(P)$ involves a function $\lambda : M \to \mathbb{N}$ such that
$$\lambda(B) < \lambda(a)$$
for every rule $a \leftarrow B \in P^M$ such that $B \subseteq M$.
- Note that $a \leftarrow B \in P^M$ and $B \subseteq M$ imply that there is a *supporting rule* $a \leftarrow B, \sim C \in \mathrm{SuppR}(P, M)$ for $a \in M$.
- However, the function λ above is not unique. E.g., the function $\lambda'(a) = \lambda(a) + 1$ satisfies this condition whenever λ does.
- In the sequel, we provide sufficient conditions for a unique *level numbering* $\lambda : M \to \mathbb{N}$ that captures the stability of M.

Level Numberings

Definition. Let M be a supported model of a normal program P. A function $\lambda : M \to \mathbb{N}$ is a *level numbering* for M iff for all $a \in M$,
$$\lambda(a) = \min\{\lambda(B) \mid a \leftarrow B, \sim C \in \mathrm{SuppR}(P, M)\}$$
where $\lambda(B) = \max\{\lambda(b) \mid b \in B\} + 1$, and in particular, $\lambda(\emptyset) = 1$.

Example. Consider a positive normal program $P = \{a \leftarrow b.\ \ b \leftarrow a.\ \}$ and its supported models $M_1 = \emptyset$ and $M_2 = \{a, b\}$:

1. There is a trivial level numbering $\lambda_1 : M_1 \to \mathbb{N}$ for M_1.
2. The requirements for a level numbering $\lambda_2 : M_2 \to \mathbb{N}$ are:
$$\lambda_2(a) = \lambda_2(b) + 1 \text{ and } \lambda_2(b) = \lambda_2(a) + 1.$$
$\Longrightarrow$ There is no such level numbering λ_2.

Properties of Level Numberings (I)

Proposition. If P is a normal program, $M \in \mathrm{SuppM}(P)$ a supported model, and λ is a level numbering for M, then $M \in \mathrm{SM}(P)$.

Proof. To prove the critical half $M \subseteq \mathrm{LM}(P^M)$ of stability, it is shown by induction on $\lambda(a)$ that $a \in M$ implies $a \in \mathrm{LM}(P^M)$.

1. If $a \in M$ has the smallest value n of $\lambda(a)$, we have $\lambda(a) = \lambda(B)$ for some $a \leftarrow B, \sim C \in \mathrm{SuppR}(P, M)$. The definition of $\lambda(B)$ implies $B = \emptyset$ and $\lambda(a) = n = 1$. Thus a is a fact in P^M and $a \in \mathrm{LM}(P^M)$.
2. For $a \in M$ such that $\lambda(a) > 1$, we note that $\lambda(a) = \lambda(B)$ for some $a \leftarrow B, \sim C \in \mathrm{SuppR}(P, M)$. It follows that $a \leftarrow B \in P^M$ and $M \models B$. The definition of $\lambda(B)$ implies $\lambda(b) < \lambda(a)$ for every $b \in B$. Thus $B \subseteq \mathrm{LM}(P^M)$ by the inductive hypothesis and $a \in \mathrm{LM}(P^M)$ holds since $a \leftarrow B \in P^M$. □

Properties of Level Numberings (II)

Proposition. A level numbering λ for $M \in \mathrm{SuppM}(P)$ is unique.

Proof. Suppose that λ is not unique, i.e., there is a different level numbering λ' for M. We prove by induction on $\lambda(a)$ that $\lambda'(a) = \lambda(a)$.

1. Suppose that $\lambda(a) = 1$. It follows that $\lambda(B) = 1$ for some $a \leftarrow B, \sim C \in \mathrm{SuppR}(P, M)$. Thus $B = \emptyset$ must be the case, and $\lambda'(B) = 1$ and $\lambda'(a) = 1$ by the definition of level numberings.
2. Then assume $\lambda(a) > 1$. The definition of $\lambda(a)$ implies that $\lambda(a) = \lambda(B)$ for some rule $a \leftarrow B, \sim C \in \mathrm{SuppR}(P, M)$. Since $\lambda(b) < \lambda(a)$ for each $b \in B$ by definition, we obtain $\lambda'(B) = \lambda(B)$ by the inductive hypothesis. Thus $\lambda'(a) \leq \lambda(a)$. Assuming $\lambda'(a) < \lambda(a)$ suggests a rule $a \leftarrow B', \sim C' \in \mathrm{SuppR}(P, M)$ with $\lambda'(B') < \lambda'(B)$ and $\lambda(B') = \lambda'(B') < \lambda(a)$, a contradiction. □

Assigning Level Numbers to Atoms

- A concrete level numbering can be obtained from the construction of the *least model* $\mathrm{LM}(P)$ for a positive program P.
- Recall that if P is finite, then $\mathrm{lfp}(\mathrm{T}_P) = \mathrm{T}_P \uparrow i$ for some $i \in \mathbb{N}$ where the operator T_P is defined by

$$\mathrm{T}_P(A) = \{a \mid a \leftarrow B \in P \text{ and } B \subseteq A\}.$$

Definition. The *level number* $\#a$ of an atom $a \in \mathrm{LM}(P)$ is the least number $n \in \mathbb{N}$ such that $a \in (\mathrm{T}_P \uparrow n) \setminus (\mathrm{T}_P \uparrow n-1)$.

Example. For a positive program consisting of

$$a. \quad a \leftarrow c. \quad b \leftarrow a. \quad c \leftarrow a, b. \quad d \leftarrow d, c.$$

we have $\mathrm{LM}(P) = \{a, b, c\}$ and the corresponding level numbers are $\#a = 1$, $\#b = 2$, and $\#c = 3$. The number $\#d$ is undefined ($d \notin \mathrm{LM}(P)$).

Properties of Level Numberings (III)

Proposition. If P is a normal program and $M \in \mathrm{SM}(P)$, then $\# : M \rightarrow \mathbb{N}$ as defined for $M = \mathrm{LM}(P^M)$ is a level numbering for M.

Proof. Now $M = \mathrm{lfp}(\mathrm{T}_{P^M})$ since $M \in \mathrm{SM}(P)$.

(i) We define $M_i = \mathrm{T}_{P^M} \uparrow i$ for $i \geq 0$.

(ii) Then the level number $\#a$ of an atom $a \in M = \mathrm{LM}(P^M)$ is the least number $i \in \mathbb{N}$ such that $a \in M_i \setminus M_{i-1}$ by definition.

(iii) Next we prove by induction on i that for each $a \in M_i$,

$$\#a = \min\{\#B \mid a \leftarrow B, \sim C \in \mathrm{SuppR}(P, M)\}$$

where $\#B = \max\{\#b \mid b \in B\} + 1$.

Proof by Induction

The base case $i = 0$ is trivial, since $M_0 = \emptyset$.

Then consider any $a \in M_i$ when $i > 0$. The case $a \in M_{i-1}$ is covered by inductive hypothesis, so let $a \in M_i \setminus M_{i-1}$. It follows that $\#a = i > 0$.

1. Now there is $a \leftarrow B, {\sim}C \in \mathrm{SuppR}(P,M)$ such that $a \leftarrow B \in P^M$ and $B \subseteq M_{i-1}$. Thus $\#B = \max\{\#b \mid b \in B\} + 1 \leq i$.
2. Assuming $\#B < i$ implies $\#b < i-1$ for all $b \in B$, $B \subseteq M_{i-2}$, and $a \in M_{i-1}$, a contradiction. Hence $\#B = i$.
3. Thus $m_a = \min\{\#B \mid a \leftarrow B, {\sim}C \in \mathrm{SuppR}(P,M)\} \leq i = \#a$.
4. Assuming $m_a < i$ implies $\#B' < i$ for some other supporting rule $a \leftarrow B', {\sim}C' \in \mathrm{SuppR}(P,M)$ and $a \in M_{i-1}$, a contradiction.

It follows that $m_a = i = \#a$ as was to be shown. $\square$

Characterization of Stable Models

Theorem. For a normal logic program P and an interpretation $M \subseteq \mathrm{Hb}(P)$, $M \in \mathrm{SM}(P)$ if and only if

$$M \in \mathrm{SuppM}(P) \text{ and there is a level numbering } \lambda \text{ for } M.$$

Example. Recall the supported models $M_1 = \emptyset$ and $M_2 = \{a,b\}$ of the normal program $P = \{a \leftarrow b.\ \ b \leftarrow a.\ \}$.

1. Now M_1 is stable since $\#_1 : M_1 \to \mathbb{N}$ is trivially a level numbering.
2. The model M_2 is not stable because the set of equations
$$\begin{cases} \#_2(a) = \#_2(b) + 1 \\ \#_2(b) = \#_2(a) + 1 \end{cases}$$
for a level numbering $\#_2$ has no solution.

2. TRANSLATION INTO ATOMIC PROGRAMS

- An *atomic* normal program $\mathrm{Tr}_{\mathrm{AT}}(P) =$
$$\mathrm{Tr}_{\mathrm{SUPP}}(P) \cup \mathrm{Tr}_{\mathrm{CTR}}(P) \cup \mathrm{Tr}_{\mathrm{MAX}}(P) \cup \mathrm{Tr}_{\mathrm{MIN}}(P)$$
is utilized as an intermediary representation.
- Level numbers have to be captured using *binary counters* which are represented by vectors $c[1\ldots n] = c_1,\ldots,c_n$ of propositional atoms.
- The logarithm $\nabla P = \lceil \log_2(|\mathrm{Hb}(P)|+2) \rceil$ gives an upper bound for the number of bits needed in such counters.
- A number of primitive operations involving binary counters of n bits are formalized as subprograms to be described next.

Primitives for Binary Counters

1. The program $\mathrm{SEL}(c[1\ldots n])$ selects a value for $c[1\ldots n]$:
$$c_1 \leftarrow {\sim}\overline{c_1}.\quad \overline{c_1} \leftarrow {\sim}c_1.\quad \ldots\quad c_n \leftarrow {\sim}\overline{c_n}.\quad \overline{c_n} \leftarrow {\sim}c_n.$$
2. The program $\mathrm{NXT}(c[1\ldots n], d[1\ldots n])$ sets the value of $d[1\ldots n]$ as the successor of the value of $c[1\ldots n]$ in binary representation.
3. The program $\mathrm{FIX}(c[1\ldots n], v)$ sets a fixed value v for $c[1\ldots n]$.
4. The program $\mathrm{LT}(c[1\ldots n], d[1\ldots n])$ checks whether the value of $c[1\ldots n]$ is lower than that of $d[1\ldots n]$.
5. The program $\mathrm{EQ}(c[1\ldots n], d[1\ldots n])$ tests whether the values of $c[1\ldots n]$ and $d[1\ldots n]$ are the same.

Remark. The activation of these primitives can be controlled with additional negative conditions.

Translation $\mathrm{Tr}_{\mathrm{SUPP}}(P)$

Definition. A rule $r = a \leftarrow B, \sim C \in P$ is translated into

$$\{a \leftarrow \sim\overline{\mathrm{bt}(r)}.\ \ \overline{\mathrm{bt}(r)} \leftarrow \sim\mathrm{bt}(r).\ \ \mathrm{bt}(r) \leftarrow \sim\overline{B}, \sim C.\ \}.$$

An atom $a \in \mathrm{Hb}(P)$ is translated into $\overline{a} \leftarrow \sim a$.

Remark. The intuitive reading of $\mathrm{bt}(r)$ is that the body of r is true.

Theorem. For a normal program P and an interpretation $M \subseteq \mathrm{Hb}(P)$, $M \in \mathrm{SuppM}(P)$ if and only if

$$N = M \cup \{\overline{a} \mid a \in \mathrm{Hb}(P) \setminus M\} \cup \{\mathrm{bt}(r) \mid r \in \mathrm{SuppR}(P,M)\} \cup \{\overline{\mathrm{bt}(r)} \mid r \in P \setminus \mathrm{SuppR}(P,M)\}$$

belongs to $\mathrm{SM}(\mathrm{Tr}_{\mathrm{SUPP}}(P))$.

Translation $\mathrm{Tr}_{\mathrm{CTR}}(P)$

- The goal of $\mathrm{Tr}_{\mathrm{CTR}}(P)$ is to select/set values for counters.
- Additional counters $\mathrm{nxt}(a)$ and $\mathrm{ctr}(r)$ of ∇P bits are associated with atoms $a \in \mathrm{Hb}(P)$ and rules $r \in P$, respectively.

Definition. An atom $a \in \mathrm{Hb}(P)$ is translated into subprograms

$$\mathrm{SEL}(a[1\ldots\nabla P], \sim\overline{a}) \text{ and } \mathrm{NXT}(a[1\ldots\nabla P], \mathrm{nxt}(a)[1\ldots\nabla P], \sim\overline{a}).$$

A rule $r = a \leftarrow B, \sim C \in P$ is translated into a subprogram

$$\begin{cases} \mathrm{FIX}(\mathrm{ctr}(r)[1\ldots\nabla P], 1, \sim\overline{\mathrm{bt}(r)}), & \text{if } B = \emptyset \text{, and} \\ \mathrm{SEL}(\mathrm{ctr}(r)[1\ldots\nabla P], \sim\overline{\mathrm{bt}(r)}), & \text{otherwise.} \end{cases}$$

Let $\mathrm{Ext}(a[1\ldots\nabla P], v)$ be the resulting set of true atoms describing the bit statuses of $a[1\ldots\nabla P]$ when the counter has a value $0 \leq v < 2^{\nabla P}$.

Translation $\mathrm{Tr}_{\mathrm{MAX}}(P)$

Definition. An atom $b \in B$ appearing in $r = a \leftarrow B, \sim C \in P$ is translated into following set of rules:

$$\mathrm{LT}(\mathrm{ctr}(r)[1\ldots\nabla P], \mathrm{nxt}(b)[1\ldots\nabla P], \sim\overline{\mathrm{bt}(r)}) \cup \mathrm{EQ}(\mathrm{ctr}(r)[1\ldots\nabla P], \mathrm{nxt}(b)[1\ldots\nabla P], \sim\overline{\mathrm{bt}(r)}) \cup \{\bot \leftarrow \sim\overline{\mathrm{bt}(r)}, \sim\overline{\mathrm{lt}(\mathrm{ctr}(r), \mathrm{nxt}(b))}\} \cup \{\mathrm{max}(r) \leftarrow \sim\overline{\mathrm{bt}(r)}, \sim\overline{\mathrm{eq}(\mathrm{ctr}(r), \mathrm{nxt}(b))}\}.$$

Moreover, a rule $r = a \leftarrow B, \sim C \in P$ is translated into

$$\bot \leftarrow \sim\overline{\mathrm{bt}(r)}, \sim\mathrm{max}(r).$$

Remark. The intuitive reading of $\mathrm{max}(r)$ is that the value of $\mathrm{ctr}(r)[1\ldots\nabla P]$ equals to the intended maximum.

Translation $\mathrm{Tr}_{\mathrm{MIN}}(P)$

Definition. A rule $r = a \leftarrow B, \sim C \in \mathrm{Def}_P(a)$ is translated into

$$\mathrm{LT}(\mathrm{ctr}(r)[1\ldots\nabla P], a[1\ldots\nabla P], \sim\overline{\mathrm{bt}(r)}) \cup \mathrm{EQ}(\mathrm{ctr}(r)[1\ldots\nabla P], a[1\ldots\nabla P], \sim\overline{\mathrm{bt}(r)}) \cup \{\bot \leftarrow \sim\overline{\mathrm{bt}(r)}, \sim\overline{\mathrm{lt}(\mathrm{ctr}(r), a)}\} \cup \{\mathrm{min}(a) \leftarrow \sim\overline{\mathrm{bt}(r)}, \sim\overline{\mathrm{eq}(\mathrm{ctr}(r), a)}\}.$$

Moreover, an atom $a \in \mathrm{Hb}(P)$ is translated into $\bot \leftarrow \sim\overline{a}, \sim\mathrm{min}(a)$.

Remark. The intuitive reading of $\mathrm{min}(a)$ is that the value of $a[1\ldots\nabla P]$ equals to the intended minimum.

Example

Recall the normal program P with rules $r_1 = a \leftarrow b$ and $r_2 = b \leftarrow a$.

- In addition to subprograms, the rules for a and r_1 are:

  $a \leftarrow \sim\overline{\mathrm{bt}(r_1)}$. $\overline{\mathrm{bt}(r_1)} \leftarrow \sim\mathrm{bt}(r_1)$. $\mathrm{bt}(r_1) \leftarrow \sim\overline{b}$. $\overline{b} \leftarrow \sim b$.

  $\bot \leftarrow \sim\overline{\mathrm{bt}(r_1)}, \sim\overline{\mathrm{lt}(\mathrm{ctr}(r_1),\mathrm{nxt}(b))}$.

  $\bot \leftarrow \sim\overline{\mathrm{bt}(r_1)}, \sim\overline{\mathrm{lt}(\mathrm{ctr}(r_1),a)}$.

  $\mathrm{max}(r_1) \leftarrow \sim\overline{\mathrm{bt}(r_1)}, \sim\overline{\mathrm{eq}(\mathrm{ctr}(r_1),\mathrm{nxt}(b))}$.

  $\mathrm{min}(a) \leftarrow \sim\overline{\mathrm{bt}(r_1)}, \sim\overline{\mathrm{eq}(\mathrm{ctr}(r_1),a)}$.

  $\bot \leftarrow \sim\overline{\mathrm{bt}(r_1)}, \sim\mathrm{max}(r_1)$. $\bot \leftarrow \sim\overline{a}, \sim\mathrm{min}(a)$.

- Rules for b and r_2 are symmetric.
- The only stable model is $N = \{\overline{a}, \overline{b}, \overline{\mathrm{bt}(r_1)}, \overline{\mathrm{bt}(r_2)}\}$.

Correctness of $\mathrm{Tr}_{\mathrm{AT}}(P)$

Definition. Let P be a normal program, $M \in \mathrm{SuppM}(P)$, $\#$ a level numbering $\# : M \rightarrow \{1,\ldots,2^{\nabla P}\}$ for M, and $e : \mathbf{2}^{\mathrm{Hb}(P)} \rightarrow \mathbf{2}^{\mathrm{Hb}(\mathrm{Tr}_{\mathrm{AT}}(P))}$ a function determined by an interpretation $e(M)$ which is the union of

1. $M \cup \{\overline{a} \mid a \in \mathrm{Hb}(P) \setminus M\}$,
2. $\{\mathrm{bt}(r) \mid r \in \mathrm{SuppR}(P,M)\} \cup \{\overline{\mathrm{bt}(r)} \mid r \in P \setminus \mathrm{SuppR}(P,M)\}$,
3. $\{\mathrm{max}(r) \mid r \in \mathrm{SuppR}(P,M)\} \cup \{\mathrm{min}(a) \mid a \in M\}$,
4. $\mathrm{Ext}(a[1\ldots\nabla P],\#a) \cup \mathrm{Ext}(\mathrm{nxt}(a)[1\ldots\nabla P],\#a+1)$ for each $a \in M$,
5. $\mathrm{Ext}(\mathrm{ctr}(r)[1\ldots\nabla P],\#B)$ where $\#B = \max\{\#b \mid b \in B\}+1$ for each $r = a \leftarrow B, \sim C \in \mathrm{SuppR}(P,M)$ and

in addition, any sets of atoms made true by comparisons involved in the subprograms $\mathrm{LT}(\ldots)$ and $\mathrm{EQ}(\ldots)$ of $\mathrm{Tr}_{\mathrm{AT}}(P)$.

Properties of $\mathrm{Tr}_{\mathrm{AT}}(P)$

Theorem. Let P be a normal program.

1. If $M \in \mathrm{SM}(P)$, then $N = e(M) \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{AT}}(P))$.
2. If $N \in \mathrm{SM}(\mathrm{Tr}_{\mathrm{AT}}(P))$, then $M = N \cap \mathrm{Hb}(P) \in \mathrm{SM}(P)$ and $N = e(M)$.

Proof. A detailed proof can be found from a research report, T. Janhunen: "*Translatability and intranslatability results for certain classes of logic programs*" [TKK/TCS, A82, 2003]. □

Corollary. For a normal program P, $P \equiv_{\mathrm{v}} \mathrm{Tr}_{\mathrm{AT}}(P)$.

Proposition. For a normal program P, the translation $\mathrm{Tr}_{\mathrm{AT}}(P)$ can be computed in time linear with respect to $||P|| \times \nabla P$.

3. REACHABILITY BENCHMARK

- The translations $\mathrm{Tr}_{\mathrm{AT}}(P)$ and $\mathrm{Comp}(P)$ are implemented as translators `lp2atomic` and `lp2sat` to be used with `lparse`.
- In the implementation, the translation $\mathrm{Tr}_{\mathrm{AT}}(P)$ was optimized in a number of ways. For instance, SCCs are fully exploited.
- The benchmark is to compute all subgraphs $\langle V_n, E\rangle$ of a directed graph $D_n = \langle V_n, E_n\rangle$ where $V_n = \{1,\ldots,n\}$,

  $$E \subseteq E_n = \{\langle i,j\rangle \mid 0 < i \leq n,\ 0 < j \leq n, \text{ and } i \neq j\},$$

  and all nodes of V_n are mutually reachable in $\langle V_n, E\rangle$.
- The experiments reported in the sequel were run on a 1.67 GHz CPU having 1GBs of main memory.

Computing All Solutions

Number of 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 S with $\|S\| > 1$	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

Computing Only One Solution

Number of Vertices	8	9	10
smodels	0.009	0.013	0.022
cmodels	0.046	0.042	0.055
lp2atomic+smodels	$>10^4$	$>10^4$	$>10^4$
lp2sat+chaff	0.771	32.6	254
lp2sat+relsat	2.51	$>10^4$	$>10^4$
wf+lp2sat+relsat	2.80	4830	$>10^4$
assat	0.023	0.028	0.037

OBJECTIVES

- You are aware of SAT solvers as potential search engines for ASP and know some systems based on this architecture:
  1. assat: http://assat.cs.ust.hk/
  2. cmodels: http://www.cs.utexas.edu/users/tag/cmodels/
  3. lp2sat: http://www.tcs.hut.fi/Software/lp2sat/
- You have tried out one of the SAT-based ASP solvers in practice.
- You know that there is a faithful and polynomial time transformation from normal programs into propositional logic.
- You are able to identify the effects of the major sources of non-modularity in the transformation.

TIME TO PONDER

Recall the characterization of a stable model $M \in \mathrm{SM}(P)$ in terms of a level numbering $\# : M \to \mathbb{N}$.

Can you think of any optimizations of $\mathrm{Tr}_{\mathrm{AT}}(P)$, e.g., when the normal program P under consideration

- contains only *binary rules* of the form $a \leftarrow B, \sim C$ where $|B| \leq 2$,
- contains only *unary rules* of the form $a \leftarrow B, \sim C$ where $|B| \leq 1$, or
- contains only *atomic rules* of the form $a \leftarrow \sim C$?

Do syntactic restrictions of this kind essentially reduce the expressive power of normal programs?