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 \( \text{Tr}_{\text{AT}} \) from normal programs into atomic programs having rules of the form \( a \leftarrow \neg C \) only.
- Such a transformation is inherently non-modular but \( \text{Tr}_{\text{AT}}(P) \) is always tight so that \( P \equiv \text{Comp}(\text{Tr}_{\text{AT}}(P)) \).
- 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 \( M \models \text{Comp}(P) \) involves a function \( \lambda : M \rightarrow \mathbb{N} \) such that
  \[
  \lambda(B) < \lambda(a)
  \]
  for every rule \( a \leftarrow B \in P^M \) such that \( B \subseteq M \).
- Note that if \( a \leftarrow B \in P^M \) and \( B \subseteq M \) imply that there is a supporting rule \( a \leftarrow B, \sim C \in \text{SuppR}(P,M) \) for \( a \in M \).
- However, the function \( \lambda \) above is not unique. E.g., the function \( \lambda'(a) = \lambda(a) + 1 \) satisfies this condition whenever \( \lambda \) does.
- In the sequel, we provide sufficient conditions for a unique level numbering \( \lambda : M \rightarrow \mathbb{N} \) that captures the stability of \( M \).

Definition. Let \( M \) be a supported model of a normal program \( P \). A function \( \lambda : M \rightarrow \mathbb{N} \) is a level numbering for \( M \) iff for all \( a \in M \),

\[
\lambda(a) = \min \{ \lambda(B) \mid a \leftarrow B, \sim C \in \text{SuppR}(P,M) \}
\]

where \( \lambda(B) = \max \{ \lambda(b) \mid b \in B \} + 1 \), and in particular, \( \lambda(0) = 1 \).

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 \( \lambda_1 : M_1 \rightarrow \mathbb{N} \) for \( M_1 \).
2. The requirements for a level numbering \( \lambda_2 : M_2 \rightarrow \mathbb{N} \) are:
   \[
   \lambda_2(a) = \lambda_2(b) + 1 \quad \text{and} \quad \lambda_2(b) = \lambda_2(a) + 1.
   \]
   \[ \Rightarrow \] There is no such level numbering \( \lambda_2 \).
### Properties of Level Numberings (I)

**Proposition.** If \( P \) is a normal program, \( M \in \text{Supp}(P) \) a supported model, and \( \lambda \) is a level numbering for \( M \), then \( M \in \text{LM}(P) \).

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

1. If \( a \in M \) has the smallest value \( n \) of \( \lambda(a) \), we have \( \lambda(a) = \lambda(B) \) for some \( a \leftarrow B, \sim C \in \text{Supp}(P,M) \). The definition of \( \lambda(B) \) implies \( B = \emptyset \) and \( \lambda(a) = n = 1 \). Thus \( a \) is a fact in \( P \) and \( a \in \text{LM}(P) \).

2. For \( a \in M \) such that \( \lambda(a) > 1 \), we note that \( \lambda(a) = \lambda(B) \) for some \( a \leftarrow B, \sim C \in \text{Supp}(P,M) \). It follows that \( a \leftarrow B \in P \) and \( M \models B \). The definition of \( \lambda(B) \) implies \( \lambda(b) < \lambda(a) \) for every \( b \in B \). Thus \( B \subseteq \text{LM}(P) \) by the inductive hypothesis and \( a \in \text{LM}(P) \) holds since \( a \leftarrow B \in P \).

\( \square \)

### Properties of Level Numberings (II)

**Proposition.** A level numbering \( \lambda \) for \( M \in \text{Supp}(P) \) is unique.

**Proof.** Suppose that \( \lambda \) is not unique, i.e., there is a different level numbering \( \lambda' \) 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 \( a \leftarrow B, \sim C \in \text{Supp}(P,M) \). 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 \( a \leftarrow B, \sim C \in \text{Supp}(P,M) \). Since \( \lambda(b) < \lambda(a) \) for each \( b \in B \) by definition, we obtain \( \lambda'(B) = \lambda(B) \) by the inductive hypothesis. Thus \( \lambda'(a) = \lambda(a) \). Assuming \( \lambda'(a) < \lambda(a) \) suggests a rule \( a \leftarrow B', \sim C' \in \text{Supp}(P,M) \) with \( \lambda'(B') < \lambda'(B) \) and \( \lambda'(B') = \lambda(B') < \lambda(a) \), a contradiction.

\( \square \)

### Properties of Level Numberings (III)

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

**Proof.** Now \( M = \text{lfp}(P_M) \) since \( M \in \text{SM}(P) \).

(i) We define \( M_i = P_M \uparrow i \) for \( i \geq 0 \).

(ii) Then the level number \( \# a \) of an atom \( a \in M = \text{LM}(P) \) 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 \( a \in M_i \),

\[ \# a = \min \{ \# B \mid a \leftarrow B, \sim C \in \text{Supp}(P,M) \} \]

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 $a \leftarrow B, \sim C \in \text{Supp}(P, M)$ 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 $m_a = \min \{#B \mid a \leftarrow B, \sim C \in \text{Supp}(P, M)\} \leq i = #a$.

4. Assuming $m_a < i$ implies $#B' < i$ for some other supporting rule $a \leftarrow B', \sim C' \in \text{Supp}(P, M)$ and $a \in M_{i-1}$, a contradiction.

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

---

Characterization of Stable Models

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

$$M \in \text{SuppM}(P)$$

and there is a level numbering $\lambda$ 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 $\#_1 : M_1 \rightarrow \mathbb{N}$ 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 $\text{Tr}_\text{AT}(P) = 

$$
\text{Tr}_\text{SUPP}(P) \cup \text{Tr}_\text{CTR}(P) \cup \text{Tr}_\text{MAX}(P) \cup \text{Tr}_\text{MIN}(P)
$$

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 $\forall P = \lceil \log_2(\|\text{Hb}(P)\| + 2) \rceil$ 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 $\text{SEL}(c[1 \ldots n])$ selects a value for $c[1 \ldots n]$:

$$c_1 \leftarrow \sim c_1. \; \ldots \; c_n \leftarrow \sim c_n. \; \ldots \; c_n \leftarrow \sim c_n.$$ 

2. The program $\text{NXT}(c[1 \ldots n], d[1 \ldots n])$ 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 $\text{FIX}(c[1 \ldots n], v)$ sets a fixed value $v$ for $c[1 \ldots n]$.

4. The program $\text{LT}(c[1 \ldots n], d[1 \ldots n])$ checks whether the value of $c[1 \ldots n]$ is lower than that of $d[1 \ldots n]$.

5. The program $\text{EQ}(c[1 \ldots n], d[1 \ldots n])$ 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.
\section*{Translation $\text{Tr}_{\text{SUPP}}(P)$}

\textbf{Definition.} A rule $r = a \leftarrow B, \neg C \in P$ is translated into

$$\{ a \leftarrow \neg \text{bt}(r). \quad \text{bt}(r) \leftarrow \neg \text{bt}(r). \quad \text{bt}(r) \leftarrow \neg B, \neg C. \}.$$  

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

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

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

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

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

\section*{Translation $\text{Tr}_{\text{CTR}}(P)$}

\textbullet \quad \text{The goal of } \text{Tr}_{\text{CTR}}(P) \text{ is to select/set values for counters.}

\textbullet \quad \text{Additional counters } \text{nxt}(a) \text{ and } \text{ctr}(r) \text{ of } \forall P \text{ bits are associated with atoms } a \in \text{Hb}(P) \text{ and rules } r \in P, \text{ respectively.}

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

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

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

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

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

\section*{Translation $\text{Tr}_{\text{MAX}}(P)$}

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

$$\begin{align*} 
\text{LT}[\text{ctr}(r)[1 \ldots \forall P], \text{nxt}(b)[1 \ldots \forall P], \neg \text{bt}(r)) \cup \\
\text{EQ}[\text{ctr}(r)[1 \ldots \forall P], \text{nxt}(b)[1 \ldots \forall P], \neg \text{bt}(r)) \cup \\
\{ \bot \leftarrow \neg \text{bt}(r), \neg \text{lt}(\text{ctr}(r), \text{nxt}(b)) \} \cup \\
\{ \max(r) \leftarrow \neg \text{bt}(r), \neg \text{eq}(\text{ctr}(r), \text{nxt}(b)) \}. 
\end{align*}$$

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

$$\bot \leftarrow \neg \text{bt}(r), \neg \max(r).$$

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

\section*{Translation $\text{Tr}_{\text{MIN}}(P)$}

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

$$\begin{align*} 
\text{LT}[\text{ctr}(r)[1 \ldots \forall P], a[1 \ldots \forall P], \neg \text{bt}(r)) \cup \\
\text{EQ}[\text{ctr}(r)[1 \ldots \forall P], a[1 \ldots \forall P], \neg \text{bt}(r)) \cup \\
\{ \bot \leftarrow \neg \text{bt}(r), \neg \text{lt}(\text{ctr}(r), a) \} \cup \\
\{ \min(a) \leftarrow \neg \text{bt}(r), \neg \text{eq}(\text{ctr}(r), a) \}. 
\end{align*}$$

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

\textbf{Remark.} The intuitive reading of $\min(a)$ is that the value of $a[1 \ldots \forall P]$ 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:
  - $a \leftarrow \neg bt(r_1)$.  $bt(r_1) \leftarrow \neg a$.  $bt_\neg \leftarrow \neg b$.
  - $\bot \leftarrow \neg bt(r_1), \neg \text{lt}(\text{ctr}(r_1), \text{nxt}(b))$.
  - $\bot \leftarrow \neg bt(r_1), \neg \text{eq}(\text{ctr}(r_1), \text{nxt}(b))$.
  - $\text{max}(a) \leftarrow \neg bt(r_1), \neg \text{eq}(\text{ctr}(r_1), a)$.
  - $\bot \leftarrow \neg bt(r_1), \neg \text{max}(r_1)$.  $\bot \leftarrow \neg a, \neg \text{min}(a)$.

- Rules for $b$ and $r_2$ are symmetric.

- The only stable model is $N = \{ \bar{a}, \bar{b}, bt(r_1), bt(r_2) \}$.

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

Theorem. Let $P$ be a normal program.

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

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

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

Definition. Let $P$ be a normal program, $M \in \text{SuppM}(P)$, # a level numbering $\#: M \rightarrow \{0, \ldots, 2^{\text{VP}}\}$ for $M$, and $\text{Hb}(P) \rightarrow 2^{\text{Hb}(\text{Tr}_{AT}(P))}$ a function determined by an interpretation $e(M)$ which is the union of

1. $M \cup \{ a \mid a \in \text{Hb}(P) \setminus M \}$,
2. $\{ \text{bt}(r) \mid r \in \text{SuppP}(P, M) \} \cup \{ \text{bt}_\neg(r) \mid r \in P \setminus \text{SuppP}(P, M) \}$,
3. $\{ \text{max}(r) \mid r \in \text{SuppP}(P, M) \} \cup \{ \text{min}(r) \mid a \in M \}$,
4. $\text{Ext}(a[1 \ldots \text{VP}], \#a) \cup \text{Ext}(\text{nxt}(a)[1 \ldots \text{VP}], \#a + 1)$ for each $a \in M$,
5. $\text{Ext}(\text{ctr}(r)[1 \ldots \text{VP}], \#B)$ where $\#B = \max\{ \#b \mid b \in B \} + 1$ for each $r = a \leftarrow B, \neg C \in \text{SuppP}(P, M)$

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

3. REACHABILITY BENCHMARK

- The translations $\text{Tr}_{AT}(P)$ and $\text{Comp}(P)$ are implemented as translators $\text{lp2atomic}$ and $\text{lp2sat}$ to be used with $\text{lpars}$.

- In the implementation, the translation $\text{Tr}_{AT}(P)$ 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 $V_n = \{1, \ldots, n\}$.

- $E \subseteq E_n = \{(i, j) \mid 0 < i \leq n, 0 < j \leq n, \text{ and } i \neq j\}$,
  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

<table>
<thead>
<tr>
<th>Number of Vertices</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>smodels</td>
<td>0.004</td>
<td>0.003</td>
<td>0.003</td>
<td>0.033</td>
<td>12</td>
</tr>
<tr>
<td>cmodels</td>
<td>0.031</td>
<td>0.030</td>
<td>0.124</td>
<td>293</td>
<td>-</td>
</tr>
<tr>
<td>lp2atomic+smodels</td>
<td>0.004</td>
<td>0.008</td>
<td>0.013</td>
<td>0.393</td>
<td>353</td>
</tr>
<tr>
<td>lp2sat+chaff</td>
<td>0.011</td>
<td>0.009</td>
<td>0.023</td>
<td>1.670</td>
<td>-</td>
</tr>
<tr>
<td>lp2sat+realsat</td>
<td>0.004</td>
<td>0.005</td>
<td>0.018</td>
<td>0.657</td>
<td>1879</td>
</tr>
<tr>
<td>wf+lp2sat+realsat</td>
<td>0.009</td>
<td>0.013</td>
<td>0.018</td>
<td>0.562</td>
<td>1598</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Models</th>
<th>1</th>
<th>1</th>
<th>18</th>
<th>1606</th>
<th>565080</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCCs (S) with (</td>
<td>S</td>
<td>&gt; 1)</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Rules (lparse)</td>
<td>3</td>
<td>14</td>
<td>39</td>
<td>84</td>
<td>155</td>
</tr>
<tr>
<td>Rules (lp2atomic)</td>
<td>3</td>
<td>18</td>
<td>240</td>
<td>664</td>
<td>1920</td>
</tr>
<tr>
<td>Clauses (lp2sat)</td>
<td>4</td>
<td>36</td>
<td>818</td>
<td>2386</td>
<td>7642</td>
</tr>
<tr>
<td>Clauses (wf+lp2sat)</td>
<td>2</td>
<td>10</td>
<td>553</td>
<td>1677</td>
<td>5971</td>
</tr>
</tbody>
</table>

### Computing Only One Solution

<table>
<thead>
<tr>
<th>Number of Vertices</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>smodels</td>
<td>0.009</td>
<td>0.013</td>
<td>0.022</td>
</tr>
<tr>
<td>cmodels</td>
<td>0.046</td>
<td>0.042</td>
<td>0.055</td>
</tr>
<tr>
<td>lp2atomic+smodels</td>
<td>&gt;10⁴</td>
<td>&gt;10⁴</td>
<td>&gt;10⁴</td>
</tr>
<tr>
<td>lp2sat+chaff</td>
<td>0.771</td>
<td>32.6</td>
<td>254</td>
</tr>
<tr>
<td>lp2sat+realsat</td>
<td>2.51</td>
<td>&gt;10⁴</td>
<td>&gt;10⁴</td>
</tr>
<tr>
<td>wf+lp2sat+realsat</td>
<td>2.80</td>
<td>4830</td>
<td>&gt;10⁴</td>
</tr>
<tr>
<td>assat</td>
<td>0.023</td>
<td>0.028</td>
<td>0.037</td>
</tr>
</tbody>
</table>

### 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/
- 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 \(M \in SM(P)\) in terms of a level numbering \(\#: M \rightarrow \mathbb{N}\).

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

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

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