1. EXPRESSIVE POWER

- In the sequel, we concentrate on the class of normal programs although many results can be generalized for smodels programs.
- It can be formally proved that normal programs under stable models are strictly more expressive than propositional theories.
- The proof is based on the existence of translations of specific kinds between normal programs and propositional theories.
- In this respect, the basic criteria imposed on a translation \( T \) are:
  1. Faithfulness: \( T \equiv \nu T \).
  2. Modularity: \( T(T_1 \cup T_2) \equiv \nu T(T_1) \cup T(T_2) \).

Here we assume that \( H_b(T) = H_b(T) \subseteq H_b(\nu T) \), i.e., \( \nu T \) may introduce new atoms which are hidden in \( T \).

Modular Representation for Clauses

- There is a faithful and modular translation \( \nu T_N \) from sets of clauses into normal programs (including constraints).

Definition

An individual clause \( A \lor \neg B \) is translated into

\[
\nu T_N(A \lor \neg B) = \{ a \leftarrow \neg \bar{a}, \bar{a} \leftarrow \neg a, \mid a \in A \lor B \} \cup \{ \bot \leftarrow B, \neg A \}.
\]

and \( \nu T_N(S) = \bigcup \{ \nu T_N(A \lor \neg B) \mid A \lor \neg B \in S \} \) for a set of clauses \( S \).

Example

The set of clauses \( S = \{ A \lor \neg B, \neg A \lor B \} \) translates into:

- \( a \leftarrow \neg \bar{a} \)
- \( \bar{a} \leftarrow \neg a \)
- \( b \leftarrow \neg B, \bar{b} \leftarrow \neg b \)
- \( \bot \leftarrow B, \neg a \)
- \( \neg b, \neg a \leftarrow a, \neg b \)
An Impossibility Result

Theorem

There is no faithful and modular translation $\text{Tr}_C$ from normal programs into sets of clauses.

Proof

Assume the contrary, i.e., for all normal programs $P$, $P_1$, and $P_2$, $P \equiv \text{Tr}_C(P)$ and $\text{Tr}_C(P_1 \cup P_2) \equiv \text{Tr}_C(P_1) \cup \text{Tr}_C(P_2)$.

Consider normal programs $P_1 = \{(a \leftarrow \sim a, \sim b, \sim c, b, c)\}$ and $P_2 = \{b\}$:

1. Now $SM(P_1) = \emptyset$ implies that $CM(\text{Tr}_C(P_1)) = \emptyset$.
2. Thus $CM(\text{Tr}_C(P_1) \cup \text{Tr}_C(P_2)) = \emptyset$ and also $CM(\text{Tr}_C(P_1 \cup P_2)) = \emptyset$.
3. It follows that $SM(P_1 \cup P_2) = \emptyset$, because $P_1 \cup P_2 \equiv \text{Tr}_C(P_1 \cup P_2)$.

A contradiction, since $SM(P_1 \cup P_2) = \{\{b\}\}$. □

Further Analysis

The preceding analysis shows that any faithful translation from normal programs into clauses is inherently non-modular.

Thus there is no chance of obtaining a transformation that would work on a rule-by-rule basis (in analogy to $\text{Tr}_N$ for clauses).

Clark’s completion is clearly non-modular because, e.g.,

$$\text{Comp}\{(a \leftarrow b, a \leftarrow \sim b, b)\} \neq \text{Comp}\{(a \leftarrow b, b)\} \cup \text{Comp}\{(a \leftarrow \sim b, b)\}.$$

The construction of $\text{LF}(P)$ is also non-modular. For instance,

$$\text{LF}\{(a \leftarrow b, b \leftarrow a)\} \neq \text{LF}\{(a \leftarrow b, b)\} \cup \text{LF}\{(a \leftarrow \sim b, b)\}.$$

These observations concretely reflect the difference between expressive powers of normal programs and propositional logic.

2. LEVEL RANKINGS AND STABILITY

Any stable model $M \in SM(P)$ is also a supported model of $P$, i.e., for every $a \in M$ there is a supporting rule $a \leftarrow B, \sim C \in \text{SuppR}(P, M)$ such that $a \leftarrow B \in P^M$ and $B \subseteq M$.

To show that a supported model $M$ of $P$ is also stable, an ordering $a_1, \ldots, a_n$ of a supported model $M$ of $P$ is needed: for each $a_i$ there is $a_i \leftarrow B_i, \sim C_i \in \text{SuppR}(P, M)$ so that $B_i \subseteq \{a_1, \ldots, a_{i-1}\}$.

By defining $\lambda(a_i) = i$, we obtain a ranking function $\lambda : M \rightarrow \mathbb{N}$: for all $a \in M$, there is $a \leftarrow B, \sim C \in \text{SuppR}(P, M)$ such that $\lambda(a) > \lambda(B) = \max\{\lambda(b) \mid b \in B\}$.

Example

Consider the program $P$: $a \leftarrow b, b \leftarrow \sim c, c \leftarrow b, a \leftarrow d, d \leftarrow a$.

The supported models of $P$ are $M_1 = \{c\}, M_2 = \{a, b, d\}$, and $M_3 = \{a, c, d\}$. The last cannot be ordered as suggested above.
Non-Uniqueness

- Ranking functions $\lambda$ introduced above are not unique.
  
  E.g., the function $\lambda'(a) = \lambda(a) + 1$ is also a ranking function whenever $\lambda$ is.
- In the sequel, we provide sufficient conditions for a level ranking $\lambda : M \to \mathbb{N}$ that captures the stability of $M$.
- If a strong level ranking $\lambda : M \to \mathbb{N}$ exists for a particular supported model $M$ of $P$, then it is unique.

Example

For the preceding program

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

and the supported model $M_2 = \{a, b, d\}$ we could set $\lambda(b) = 1$, $\lambda(a) = 2$, and $\lambda(d) = 3$ in order to obtain a unique level ranking.

Properties of Level Rankings (I)

Proposition

If $P$ is a normal program, $M \in \text{Supp}(P)$ a supported model, and $\lambda$ a level ranking of $M$ for $P$, then $M \in \text{SM}(P)$.

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 ranking $\lambda_1 : M_1 \to \mathbb{N}$ for $M_1$.
2. There is no level ranking $\lambda_2 : M_2 \to \mathbb{N}$ because $\lambda_2(a) \geq \lambda_2(b) + 1$ and $\lambda_2(b) \geq \lambda_2(a) + 1$ cannot be satisfied simultaneously.

Assigning Level Ranks to Atoms

- A concrete level ranking can be obtained from the construction of the least model $\text{LM}(P)$ for a positive program $P$.
- Recall that if $P$ is finite, then $\text{lp}(T_P) = T_P \uparrow i$ for some $i \in \mathbb{N}$ where $T_P$ is defined by $T_P(A) = \{a \mid a \leftarrow B \in P \text{ and } B \subseteq A\}$.

Definition

The level rank $\# a$ of an atom $a \in \text{LM}(P)$ is the least number $n \in \mathbb{N}$ such that $a \in (T_P \uparrow n) \setminus (T_P \uparrow n - 1)$.

Example

For a positive program consisting of

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

we have $\text{LM}(P) = \{a, b, c\}$ and the corresponding level ranks are $\# a = 1$, $\# b = 2$, and $\# c = 3$. But $\# d$ is undefined, as $d \not\in \text{LM}(P)$.
Properties of Level Rankings (II)

3. POLYNOMIAL TRANSLATION

Characterization of Stable Models

Theorem
For a normal logic program \( P \) and an interpretation \( M \subseteq Hb(P) \), \( M \in SM(P) \) if and only if \( M \in SuppM(P) \) and there is a level ranking \( \lambda \) of \( M \) for \( P \).

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 ranking.
2. The model \( M_2 \) is not stable because the set of equations

\[
\begin{align*}
\#_2(a) &\geq \#_2(b) + 1 \\
\#_2(b) &\geq \#_2(a) + 1 
\end{align*}
\]

imposed on a potential level ranking \( \#_2 \) has no solution.

Strong Level Rankings

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

1. There is a rule \( a \leftarrow B, \sim C \in SuppR(P, M) \) such that for every \( b \in B \), \( \lambda(a) \geq \lambda(b) + 1 \).
2. If there is an atomic rule \( a \leftarrow \sim C \in SuppR(P, M) \), then \( \lambda(a) = 1 \).
3. For every rule \( a \leftarrow B, \sim C \in SuppR(P, M) \) such that \( B \neq \emptyset \), there is some \( b \in B \) such that \( \lambda(b) + 1 \geq \lambda(a) \).

Theorem
Let \( M \) be a supported model of a normal program \( P \).

1. There is a strong level ranking of \( M \) for \( P \) iff \( M \in SM(P) \).
2. If there is a strong level ranking \( \lambda \) of \( M \) for \( P \), then \( \lambda \) is unique.

3. POLYNOMIAL TRANSLATION

- SAT solvers provide a promising computational platform for logic programs under the stable model semantics.
- A number of systems exploiting native SAT technology exist: assat, cmodels, lp2sat, ...
- The computation of stable models presumes a translation of a ground program into clauses (in DIMACS or via APIs).
- The translations exploited by the systems mentioned above vary as regards compactness and volatility.
- Any faithful translation from programs to sets of clauses is known to be inherently non-modular.

\[
\begin{align*}
\{a \leftarrow \sim b. \ b \leftarrow \sim a. \} &\implies \{a \lor b, \sim a \lor \sim b\} \\
\{a \leftarrow b. \ b \leftarrow a. \} &\implies \{a \lor \sim b, \sim a \lor b\} \cup \{\sim a \lor \sim b\}
\end{align*}
\]
Existing Translations

The approach presented here is based on the following translations:

- The translation of normal programs into propositional clauses [T. Janhunen, ECAI, 2004] of length $O(||P|| \times \log_2 |Hb(P)|)$.
- The translation of normal programs into difference logic [I. Niemelä, AMAI, 2008] of length $O(||P||)$.

In particular, the new translation
1. covers also extended rule types such as choice rules, cardinality rules, and weight rules [Simons, LPNMR, 1999];
2. exploits level rankings rather than level numberings; and
3. removes the asymmetry of positive and negative subgoals.

Stepwise Translation

The translation of smodels programs is feasible in three steps:

1. Translate away cardinality and weight rules as well as choice rules under stable model semantics.
2. Capture stable models with supported models.
3. Apply Clark's completion and clausify in Tseitin style.

Effects of Normalization

The linear translation of choice rules into normal rules was presented when extended rule types were introduced.

The normalization of weight rules can be based on the primitives used to normalize cardinality rules.

Moreover, weight rules can be simplified in a number of ways before performing actual normalization.

Normalizing Cardinality Constraints

- We adapt Eén and Sörensson's translation [2006] of pseudo-Boolean constraints for cardinality constraints.
- The preservation of positive dependencies is crucial.

Example

The rule $a \leftarrow 3 \{b_1, b_2, b_3, \sim c_1, \sim c_2\}$ is captured by:

\[
\begin{align*}
a & \leftarrow \cnt(3,1) \leftarrow \cnt(3,2) \leftarrow \cnt(3,3) \\
& \uparrow b_1 \uparrow b_2 \uparrow b_3 \\
& \cnt(2,2) \leftarrow \cnt(2,3) \leftarrow \cnt(2,4) \\
& \uparrow b_2 \uparrow b_3 \uparrow \sim c_1 \\
& \cnt(1,3) \leftarrow \cnt(1,4) \leftarrow \cnt(1,5) \\
& \uparrow b_3 \uparrow \sim c_1 \uparrow \sim c_2
\end{align*}
\]

An example of a full command line:

```bash
$ gringo prog.lp | lp2normal2 | lp2lp2 | lp2sat | precosat
```

\[\text{Input} \quad \text{Output} \quad \text{Semantics} \quad \text{Tool}\]

\[
\begin{array}{llll}
\text{smodels program } P & \text{Normal}(P) & \text{SM(Normal}(P)) & \text{lp2normal2} \\
\text{Normal program } P & \text{LP2LP}(P) & \text{SuppM(LP2LP}(P)) & \text{lp2lp2} \\
\text{Normal program } P & \text{Comp}(P) & \text{CM(Comp}(P)) & \text{lp2sat}
\end{array}
\]
Representing Level Rankings

- We assume that all rules have been normalized.
- A vector $a_1 \ldots a_n$ of new atoms represents the level rank of an atom $a \in H_b(P)$ in a binary representation.
- Moreover, new atoms are introduced to capture the results of comparing level ranks against each other.
- For instance, given two atoms $a$ and $b$, the following subprograms for $a$:
  - $\text{It}(b,a)$ captures the condition $#b < #a$ and
  - $\text{succ}(b,a)$ captures $#a = #b + 1$.
- The respective subprograms $\text{LT}_n(b,a)$ and $\text{Succ}_n(b,a)$ that define these new atoms have to be introduced.
- The translation respects the strongly connected components of $DG^+(P)$ for the input program $P$.

Primitives for Binary Counters

Let $a$ and $b$ be atoms in the program subject to translation.

1. The program $\text{SEL}_n(a)$ selects a level rank $0 \leq #a < 2^n$ for the atom $a$.
2. The role of $\text{CLR}_n(a)$ is to clear the rank $#a$, i.e., to assign $#a = 0$ for the atom $a$.
3. The program $\text{LT}_n(a,b)$ checks whether $#a < #b$.
4. The program $\text{Succ}_n(a,b)$ checks whether $#b = #a + 1$ holds.

Example

When $n = 2$ we obtain, e.g., the following subprograms for $a$:

- $\text{SEL}_2(a) \leftrightarrow a_1 \leftrightarrow \sim a_1$.  
- $\text{CLR}_2(a) \leftrightarrow a_2 \leftrightarrow \sim a_2$.

Capturing Stability with Supported Models

Let $a$ be an atom in a program $P$ and $S$ the respective SCC of $DG^+(P)$.

1. Each rule $r = a \leftarrow B, \sim C$ is split into two normal rules $a \leftarrow b\text{(r)}$, $b\text{(r)} \leftarrow B, \sim C$.
2. For each rule $r = a \leftarrow B, \sim C$ such that $B \cap S = \emptyset$, introduce a rule for external support: $\text{ext}(a) \leftarrow b\text{(r)}$.
3. Select $\#a$ using $\text{SEL}_n(a)$ and assign $\#a = 0$ using $\text{CLR}_n(a) \leftarrow \sim a$ and $\text{CLR}_n(a) \leftarrow \text{ext}(a)$.
4. For each rule $r = a \leftarrow B, \sim C$ such that $B \cap S = \{b_1, \ldots, b_m\} \neq \emptyset$, introduce a rule for internal support: $\text{int}(a) \leftarrow b\text{(r)}$, $\text{It}(b_1,a), \ldots, \text{It}(b_m,a)$.

Weak Ranking Constraints

- The rules above essentially capture weak ranking constraints.
- The respective translation is denoted by $\text{LP2LP}^w(P)$.
- This translation is not yet strict enough in view of a bijective correspondence of models (as insisted by $\equiv_v$).

Theorem

Let $P$ be a normal logic program.

1. If $M \in \text{SM}(P)$, then there is $N \in \text{SuppM}(\text{LP2LP}^w(P))$ so that $M = N \cap H_b(P)$.
2. If $N \in \text{SuppM}(\text{LP2LP}^w(P))$, then $M \in \text{SM}(P)$ for $M = N \cap H_b(P)$.
Enhanced Translation

- Level numberings based on weak ranking constraints allow for gaps in the overall numberings.
  E.g., for \{a. b ← a.\} one could assign \#a = 1 and \#b = 3.
- The gaps can be excluded using strong ranking constraints captured by constrains of the form given below.
- The resulting translation \(\text{LP2LP}^w(P)\) extends \(\text{LP2LP}^w(P)\).

6. For each rule \(r = a ← B, ∼C\) such that \(B ∩ S = \{b_1, ..., b_m\} \neq \emptyset\), include the strong ranking constraint

\[ ∼b(1, a)_1, ∼\text{succ}(b_1, a)_n, ..., \text{lt}(b_m, a)_1, ∼\text{succ}(b_m, a)_n. \]

and subprograms

\[ \text{SUCC}_n(b_1, a) ← b(r), ..., \text{SUCC}_n(b_m, a) ← b(r). \]

Overall Correctness

Theorem
Let \(P\) be a normal logic program.

1. If \(M \in \text{SM}(P)\), then there is a unique model \(N \in \text{SM}(\text{LP2LP}^w(P))\) such that \(M = N ∩ \text{Hb}(P)\).
2. If \(N \in \text{SM}(\text{LP2LP}^w(P))\), then \(M \in \text{SM}(P)\) for \(M = N ∩ \text{Hb}(P)\).

Corollary
Let \(P\) be an smodels program and \(S = \text{Comp}(\text{LP2LP}^w(\text{Normal}(P)))\).

1. If \(M \in \text{SM}(P)\), then there is a unique interpretation \(N \subseteq \text{Hb}(S)\) such that \(N \models S\) and \(M = N ∩ \text{Hb}(P)\).
2. If \(N \models S\) for \(N \subseteq \text{Hb}(S)\), then \(M \in \text{SM}(P)\) for \(M = N ∩ \text{Hb}(P)\).

OBJECTIVES

- You understand the difference of normal logic programs and propositional logic in terms of expressive power.
- You are able to define desirable properties for translations:
  - faithfulness,
  - modularity, and
  - polynomiality (even linearity).
- You know the two main sources of non-modularity when translating ground logic programs into propositional clauses.
- You are aware of SAT solvers as potential search engines for ASP and know some systems based on this architecture:
  1. assat: assat.cs.ust.hk
  2. cmodels: www.cs.utexas.edu/users/tag/cmodels
  3. lp2sat: research.ics.aalto.fi/software/asp/lp2sat
- You know that there is a faithful and polynomial (subquadratic) time transformation from normal programs into propositional logic.

TIME TO PONDER

Recall the characterization of a stable model \(M \in \text{SM}(P)\) in terms of a level ranking \(\#: M \rightarrow \mathbb{N}\).

Can you think of any optimizations of \(\text{LP2LP}(P)\), e.g., when the normal program \(P\) under consideration

- contains only binary rules of the form \(a ← B, ∼C\) where \(|B| ≤ 2\),
- contains only unary rules of the form \(a ← B, ∼C\) where \(|B| ≤ 1\),
  or
- contains only atomic rules of the form \(a ← C\)?

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