Lloyd-Topor Completion
and General Stable Models

Vladimir Lifschitz and Fangkai Yang
Department of Computer Science
The University of Texas at Austin
AAAII Spring Symposium on Knowledge Representation, Reasoning and Robotics

- AAAI Spring Symposium on Knowledge Representation, Reasoning and Robotics
- Mar 24 – 26, 2014, Palo Alto, USA
- Organizers
  - Esra Erdem (Sabanci University)
  - Volkan Patoglu (Sabanci University)
  - Mohan Sridharan (Texas Tech University)
  - Fangkai Yang (University of Texas at Austin)
- Invited speaker includes Michael Gelfond (Texas Tech University).
(Ferraris et al, 2011) defines general stable models.

The new definition proposes an operator $SM$ that turns a first-order sentence $F$ w.r.t a set of intensional predicates $p$ into a stronger second order sentence $SM_p[F]$. The stable model of $F$ relative to $p$ is the arbitrary models of $SM_p[F]$

Logic program without strong negation can be identified as sentences, so that $SM$ is applicable to logic program as well.

Question: can we use program completion to characterize general stable models?

Since the definition of general stable models refers to programs that may contain quantifiers, we will use the generalization of completion from (Lloyd & Topor, 1978).

Our results can be applied to program more general than the rules used in (Lloyd & Topor, 1978).
Lloyd-Topor Program, Choice Rules and Constraints

• A Lloyd-Topor rule is a rules of the form

\[ p(t) \leftarrow G. \]

It can be identified as formula \( \forall(G \rightarrow p(t)) \).

• A constraint is a rule of the form

\[ \bot \leftarrow G, \]

identified as \( \neg G \).

• A choice rule is a rule of the form

\[ p(x) \lor \neg p(x) \leftarrow G. \]

It is abbreviated as

\[ \{p(x)\} \leftarrow G. \]

and can be identified as \( \forall(G \rightarrow p(x) \lor \neg p(x)) \).
(Ferraris et al, 2011) defines operator $SM$ that turns a first order sentence $F$ into a stronger second order sentence.

- The *stable models* of $F$ are arbitrary models of $SM[F]$.
- Logic programs are identified with first order sentences. Example:

\[
p(a), \\
q(b), \\
p(x) \leftarrow q(x)
\]

is viewed as sentence $F$:

\[
p(a) \land q(b) \land \forall x(q(x) \rightarrow p(x)),
\]

and $SM[F]$ is

\[
p(a) \land q(b) \land \forall x(q(x) \rightarrow p(x)) \land \text{some second order sentence}.
\]
A Lloyd-Topor program is a finite set of rules of the form

\[ p(t) \leftarrow G \]

Let \( \Pi \) be a Lloyd-Topor program, and \( p \) a predicate constant (other than equality). Let

\[ p(t^i) \leftarrow G^i \quad (i = 1, 2, \ldots) \]

be all rules of \( \Pi \) that contain \( p \) in the head. The completed definition of \( p \) is the formula

\[ \forall x \left( p(x) \leftrightarrow \bigvee_i \exists y^i (x = t^i \land G^i) \right) \]

where \( x \) is a list of distinct variables not appearing in any of the rules (1), and \( y^i \) is the list of free variables in (1).

The completion of \( \Pi \), denoted by \( \text{Comp}[\Pi] \), is the conjunction of the completed definitions of all predicate constants.
Example

The completion of the program

\[
p(a), \quad q(b), \\
p(x) \leftarrow q(x)
\]

is

\[
\forall x_1 (p(x_1) \leftrightarrow x_1 = a \lor \exists x (x_1 = x \land q(x))), \\
\forall x_1 (q(x_1) \leftrightarrow x_1 = b),
\]

which is equivalent to

\[
\forall x (p(x) \leftrightarrow x = a \lor q(x)), \\
\forall x (q(x) \leftrightarrow x = b).
\]
(Ferraris et al, 2011) extends the definition of the predicate dependency graph of a logic program to arbitrary first-order sentences. The vertices of the graph are predicate symbols. A formula is tight if its predicate dependency graph is acyclic.

From the results of the paper it follows that for a tight Lloyd-Topor program \( \Pi \), \( \text{SM}[\Pi] \) is equivalent to \( \text{Comp}[\Pi] \).

For instance, the predicate dependency graph of program \( \Pi_1 \)

\[
p(a), \quad q(b), \\
p(x) \leftarrow q(x),
\]

contains only one edge \( p \rightarrow q \) and is acyclic. So \( \text{SM}[\Pi_1] \) is equivalent to \( \text{Comp}[\Pi_1] \).
Need for Generalization

- Consider program $\Pi_2$ containing a single rule

$$p(a) \leftarrow p(x) \land x \neq a.$$ 

We will see that $\text{SM}[\Pi_2]$ is equivalent to $\text{Comp}[\Pi_2]$. But this program is not tight according to (Ferraris et al, 2011).

- Instead of the predicate dependency graph, (Lee & Meng, 2011) use the first order dependency graph. It allows them to define *atomic tightness*, which is more general than tightness. But this program is not even atomic-tight.

- Similar examples appear in applications of ASP to knowledge representation.
Conditional Equivalence between SM and Comp

- Consider program $\Pi_3$

\[
\begin{align*}
  p(x) & \leftarrow q(x), \\
  q(a) & \leftarrow p(b).
\end{align*}
\]

- $\text{SM}[\Pi_3]$ is stronger than its completion $\text{Comp}[\Pi_3]$

\[
\begin{align*}
  \forall x (p(x) \leftrightarrow q(x)), \\
  \forall x (q(x) \leftrightarrow x = a \land p(b)).
\end{align*}
\]

- However,

\[
a \neq b \models \text{SM}[\Pi_3] \leftrightarrow \text{Comp}[\Pi_3].
\]

It follows that the stable models of

\[
\Pi_3 \cup \{ \leftarrow a = b \}
\]

are characterized by the first order formula $\text{Comp}[\Pi_3] \land a \neq b$. 
Auxiliary Definitions

- A subformula $G$ of a first order formula is *positive* if the number of implications containing $G$ in the antecedent is even. ($\neg F$ stands for $F \rightarrow \bot$).

- A subformula $G$ is *nonnegated* if it does not belong to a subformula of the form $\neg F'$. 
The rule dependency graph of a Lloyd-Topor program II is the digraph that has

- rules of II, with variables renamed arbitrarily, as its vertices, and
- an edge from a rule \( p(t) \leftarrow G \) to a rule \( p'(t') \leftarrow G' \), labeled by an atomic formula \( p'(s) \), if \( p'(s) \) has a positive nonnegated occurrence in \( G \).

Example: the rule dependency graph of program

\[
p(a, b),
q(x, y) \leftarrow p(y, x) \land \neg p(x, y)
\]

has the edges

\[
q(x_1, y_1) \leftarrow p(y_1, x_1) \land \neg p(x_1, y_1) \\
\downarrow p(y_1, x_1) \\
p(a, b)
\]

for arbitrary pairs of distinct variables \( x_1 \) and \( y_1 \).
Chains

A finite path in the rule dependency graph of II is a *chain* if the rules at its vertices have no common variables. Example: for the program with the rules

\[
\begin{align*}
p(x) & \leftarrow q(x) \\
q(x) & \leftarrow r(x) \\
r(x) & \leftarrow s(x)
\end{align*}
\]

chains of length 2 have the form

\[
\begin{align*}
p(x_1) & \leftarrow q(x_1) \\
\downarrow q(x_1) \\
q(x_2) & \leftarrow r(x_2) \\
\downarrow r(x_2) \\
r(x_3) & \leftarrow s(x_3)
\end{align*}
\]

where \(x_1, x_2, x_3\) are pairwise distinct variables.

**Fact.** A Lloyd-Topor program II is tight iff there exists \(n\) such that II has no chains of length \(n\).
Let $C$ be a chain

\[ p_0(t^0) \leftarrow Body_0 \]
\[ \downarrow p_1(s^1) \]
\[ p_1(t^1) \leftarrow Body_1 \]
\[ \downarrow p_2(s^2) \]
\[ \ldots \ldots \ldots \]
\[ \downarrow p_n(s^n) \]
\[ p_n(t^n) \leftarrow Body_n \]

in a Lloyd-Topor program. The corresponding \textit{chain formula} $F_C$ is the conjunction

\[ \left( \bigwedge_{i=1}^{n} s^i = t^i \right) \land \left( \bigwedge_{i=0}^{n} Body_i \right). \]
Example

If $C$ is

\[
q(x_1, y_1) \leftarrow p(y_1, x_1) \land \neg p(x_1, y_1)
\]

\[
\downarrow p(y_1, x_1)
\]

\[
p(a, b)
\]

then the chain formula $F_C$ is

\[
(y_1 = a \land x_1 = b) \land (p(y_1, x_1) \land \neg p(x_1, y_1)).
\]
Main Theorem

Let \( \Gamma \) be a set of sentences. About a Lloyd-Topor program \( \Pi \) we will say that it is \textit{tight relative to} \( \Gamma \), or \( \Gamma \)-tight, if there exists a positive integer \( n \) such that, for every chain \( C \) in \( \Pi \) of length \( n \),

\[
\Gamma, \text{Comp}[\Pi] \models \forall \neg F_C.
\]

Any tight program is trivially \( \Gamma \)-tight for any \( \Gamma \).

Main Theorem. \textit{If a Lloyd-Topor program} \( \Pi \) \textit{is} \( \Gamma \)-tight then

\[
\Gamma \models \text{SM}[\Pi] \iff \text{Comp}[\Pi].
\]
Example

Consider program $\Pi_2$:

$$p(a) \leftarrow p(x) \land a \neq x.$$ 

It is tight relative to $\emptyset$. Indeed, any chain of length 1 has the form

$$p(a) \leftarrow p(x_1) \land a \neq x_1$$

$$\downarrow p(x_1)$$

$$p(a) \leftarrow p(x_2) \land a \neq x_2,$$

and the chain formula

$$x_1 = a \land p(x_1) \land a \neq x_1 \land p(x_2) \land a \neq x_2$$

is contradictory. So $\mathcal{V}F_C$ is logically valid for any chain $C$ of length 1.
Logic programs can be used to describe effects of actions. Consider logic program $M$ describing moving objects from one location to another.

- **Facts**

  \[
  \text{step}(0), \text{step}(1), \ldots, \text{step}(\hat{k}); \\
  \text{next}(\hat{0}, \hat{1}), \text{next}(\hat{1}, \hat{2}), \ldots, \text{next}(\hat{k}-1, \hat{k});
  \]

- **Rules**

  \[
  \{ \text{at}(x, y, 0) \} \leftarrow \text{object}(x) \land \text{place}(y) \\
  \text{at}(x, y, t_2) \leftarrow \text{move}(x, y, t_1) \land \text{next}(t_1, t_2) \\
  \{ \text{at}(x, y, t_2) \} \leftarrow \text{at}(x, y, t_1) \land \text{next}(t_1, t_2)
  \]

describe initial states, effects of actions, and the commonsense law of inertia.
Moving Objects (2)

- Unique name constraints
  \[ \hat{i} = \hat{j} \quad (1 \leq i < j \leq k) \]

- Constraints describing the arguments of \textit{at} and \textit{move}
  \[
  \leftarrow \text{at}(x, y, z) \land \neg (\text{object}(x) \land \text{place}(y) \land \text{step}(z))
  \]
  \[
  \leftarrow \text{move}(x, y, z) \land \neg (\text{object}(x) \land \text{place}(y) \land \text{step}(z))
  \]

- The uniqueness of location constraint
  \[ \leftarrow \text{at}(x, y_1, z) \land \text{at}(x, y_2, z) \land y_1 \neq y_2 \]

- The existence of location constraint
  \[ \leftarrow \text{object}(x) \land \text{step}(z) \land \neg \exists y \; \text{at}(x, y, z). \]

Intensional predicates: \( p = \{ \text{next}, \text{step}, \text{at} \}. \)
Characterizing the Stable Models of $M$

$H$: the conjunction of the constraints from $M$ written as first order sentences.

**Proposition** $\text{SM}_p[M]$ is equivalent to the conjunction of $H$ with the universal closures of the formulas

$$\text{step}(z) \leftrightarrow \bigvee_{i=0}^{k} z = \hat{i},$$

$$\text{next}(z, u) \leftrightarrow \bigvee_{i=0}^{k-1} (z = \hat{i} \land u = \hat{i+1}),$$

$$\text{at}(x, y, \hat{i+1}) \leftrightarrow (\text{move}(x, y, \hat{i}) \lor (\text{at}(x, y, \hat{i}) \land \neg \exists w \text{ move}(x, w, \hat{i})))$$

$$(i = 0, \ldots, k - 1).$$
Plan of the Proof (1)

To use Main Theorem, we will consider Lloyd-Topor program \( \Pi_4 \):

\[
\begin{align*}
\text{step}(0), \text{step}(1), \ldots \text{step}(k); \\
\text{next}(\hat{0}, \hat{1}), \text{next}(\hat{1}, \hat{2}), \ldots, \text{next}(\hat{k-1}, \hat{k}); \\
\text{at}(x, y, t_2) & \leftarrow \text{move}(x, y, t_1) \land \text{next}(t_1, t_2) \\
\text{at}(x, y, 0) & \leftarrow \text{object}(x) \land \text{place}(y) \land \neg\neg\text{at}(x, y, 0) \\
\text{at}(x, y, t_2) & \leftarrow \text{at}(x, y, t_1) \land \text{next}(t_1, t_2) \land \neg\neg\text{at}(x, y, t_2) \\
\text{object}(x) & \leftarrow \neg\neg\text{object}(x), \\
\text{place}(y) & \leftarrow \neg\neg\text{place}(y), \\
\text{move}(x, y, z) & \leftarrow \neg\neg\text{move}(x, y, z).
\end{align*}
\]

From the results of (Ferraris et al, 2011) it easily follows that \( SM_p[M] \) is equivalent to \( SM[\Pi_4] \land H \).
Plan of the Proof (2)

**Fact** Program $\Pi_4$ is $H$-tight.

By Main Theorem, it follows that

$$H \models SM[\Pi_4] \leftrightarrow \text{Comp}[\Pi_4].$$

Consequently

$$SM_p[M] \leftrightarrow SM[\Pi_4] \land H \leftrightarrow \text{Comp}[\Pi_4] \land H.$$
Conclusion

- We proposed a new method for representing $SM[F]$ in the language of first-order logic. It is more general than the methods of (Ferraris et al., 2011). Relationship with (Lee & Meng, 2011) requires further study.

- This method allows us to prove the equivalence between some ASP descriptions of dynamic domains to axiomatizations based on successor state axioms.

- Proof of Main Theorem is based on the theory of stable models of infinitary propositional formulas developed by Mirek Truszczynski.