Implementing Ordered Disjunction Using Answer Set Solvers for Normal Programs

G. Brewka, I. Niemela, T. Syrjanen

August 1, 2003

Talk Outline

- $\Rightarrow$ **Syntax and Answer Sets of Logic Programs with Ordered Disjunction**
- Preferred Answer Sets
- Implementation
- An Application and Conclusions

Syntax

- Logic Programming with Ordered Disjunction (LPOD) is an extension of logic programming with classical and default negation.
- Rules are of the form:

  $h_1 \times h_2 \times \ldots \times h_k \leftarrow l_1, \ldots, l_m, \text{not } l_{m+1}, \ldots, \text{not } l_n$

  where h's and l's are literals.
- Intuitive reading: if the body is satisfied, then believe h_1 *if possible*, otherwise believe h_2 *if possible*, ..., otherwise believe h_k.

Split Programs

Definition 1 Let r be a rule $h_1 \times \ldots \times h_k \leftarrow \Gamma$. For $i \leq k$ we define the i^{th} *option of* r, r^i, as:

$$h_i \leftarrow \Gamma, \text{not } h_1, \ldots, \text{not } h_{i-1}.$$

Definition 2 Let P be a LPOD. P' is a *split program of* P if it is obtained from P by replacing each rule in P by one of its options.

Example 1 Program P_1:

$$\begin{array}{lcl} a \times b & \leftarrow & \text{not } c. \\ b \times c & \leftarrow & \text{not } d. \end{array}$$

has 4 split programs:

$$\begin{array}{lcl} a & \leftarrow & \text{not } c. \\ b & \leftarrow & \text{not } d. \end{array} \qquad \begin{array}{lcl} b & \leftarrow & \text{not } c, \text{not } a. \\ b & \leftarrow & \text{not } d. \end{array}$$

$$\begin{array}{lcl} a & \leftarrow & \text{not } c. \\ c & \leftarrow & \text{not } d, \text{not } b. \end{array} \qquad \begin{array}{lcl} b & \leftarrow & \text{not } c, \text{not } a. \\ c & \leftarrow & \text{not } d, \text{not } b. \end{array}$$

Answer Sets

Definition 3 Let P be a LPOD. A set of literals A is an answer set of P if it is an answer set of a split program, P' of P.

Hence, program P_1 above has 3 answer sets: $\{a, b\}$, $\{c\}$, $\{b\}$.

Example 2 Program:

$$\begin{array}{rcl} a \times b & \leftarrow & \text{not } c. \\ b & \leftarrow & a. \end{array}$$

has two answer sets: $\{a, b\}$ and $\{b\}$.

Example 3 Program:

$$\begin{array}{rcl} a \times b & \leftarrow & \text{not } c. \\ a & \leftarrow & b. \end{array}$$

has one answer set: $\{a\}$.

Talk Outline

- Syntax and Answer Sets of Logic Programs with Ordered Disjunction

$\Rightarrow$ **Preferred Answer Sets**

- Implementation
- An Application and Conclusions

Degrees of Satisfaction

Degrees of satisfaction are used to distinguish between more and less intended answer sets of a LPOD.

Definition 4 Let A be an answer set of a LPOD P and r be a rule $h_1 \times \ldots \times h_k \leftarrow \Gamma$. We say that:

- *A satisfies r to degree* 1 if A does not satisfy Γ.
- *A satisfies r to degree j ($1 \leq j \leq k$)* if A satisfies Γ and $j = min\{i \mid h_i \in A\}$.

The degree of r in A is denoted by $deg_A(r)$.

Degrees of satisfaction are intended as penalties.

Definition 5 Let P be a LPOD and A a set of literals. The set of rules of P that satisfy A to degree i is denoted by $A^i(P)$. In other words:

$$A^i(P) = \{r \in P \mid deg_A(r) = i\}.$$

Preference Criteria

Definition 6 Let A_1 and A_2 be answer sets of a LPOD P. A_1 is *cardinality-preferred* to A_2 ($A_1 >_c A_2$) iff there is i such that $|A_1^i(P)| > |A_2^i(P)|$, and for all $j < i$, $|A_1^j(P)| = |A_2^j(P)|$.

Definition 7 Let A_1 and A_2 be answer sets of a LPOD P. A_1 is *inclusion-preferred* to A_2 ($A_1 >_i A_2$) iff there is i such that $A_1^i(P) \supset A_2^i(P)$, and for all $j < i$, $A_1^j(P) = A_2^j(P)$.

Definition 8 Let A_1 and A_2 be answer sets of a LPOD P. A_1 is *Pareto-preferred* to A_2 ($A_1 >_p A_2$) iff there is $r \in P$ such that $deg_{A_1}(r) < deg_{A_2}(r)$, and for no $r' \in P$ $deg_{A_1}(r') > deg_{A_2}(r')$.

Definition 9 A set of literals A is a *x-preferred answer set of a LPOD P* (where $x \in \{c, i, p\}$) iff A is an answer set of P and there is no answer set A' of P such that $A' >_x A$.

Preference Criteria (cont.)

Example 4 Consider program P_2 that performs the choice of a hotel:

$$dist(walking) \times \neg dist(walking).$$
$$stars(3) \times stars(2).$$
$$\leftarrow dist(walking), stars(3).$$

P_2 has two x-preferred answer sets:
$S_1 = \{dist(walking), stars(2)\}$, and
$S_2 = \{\neg dist(walking), stars(3)\}$.

Now consider program P_3:

$$dist(walking) \times \neg dist(walking).$$
$$stars(4) \times stars(3) \times stars(2).$$
$$\leftarrow dist(walking), stars(3).$$
$$\leftarrow stars(4).$$

S_1 is cardinality-preferred and inclusion-preferred to S_2, but none of them is Pareto-preferred to the other. Hence, P_3 has two p-preferred answer sets and one c-preferred and i-preferred answer set.

Talk Outline

- Syntax and Answer Sets of Logic Programs with Ordered Disjunction
- Preferred Answer Sets

$\Rightarrow$ **Implementation**

- An Application and Conclusions

Algorithm

The algorithm to compute the x-preferred answer sets of a LPOD P is based on two smodels programs:

- the *generator* $G(P)$ which computes all the answer sets of P, and
- the *tester* $T_x(P, M)$ which checks whether answer set M of P is x-preferred.

Algorithm (for a given cardinality criterion x):

1. use $G(P)$ to compute one answer set, M, of P
2. use $T_x(P, M)$ to find an answer set, M', such that $M' >_x M$
3. if no such M' exists, return M
4. otherwise, repeat from step 1

Note: if we are finding only one preferred answer set, step 4 can be replaced by:

4′. otherwise, set $M = M'$ and repeat from step 2

Generator Program $G(P)$

$G(P)$ essentially encodes all possible split programs of P by adding an explicit choice over the options of each ordered disjunction.

Definition 10 Let P be a LPOD and $r = h_1 \times \ldots \times h_k \leftarrow \Gamma$ be a rule from P. We define the following functions:

% The translation, $G(r,i)$, of the i^{th} option of r

$$G(r,i) = \{h_i \leftarrow c(r,i), \text{not } h_1, \ldots, \text{not } h_{i-1}, \Gamma\} \cup \{\leftarrow h_i, \text{not } c(r,i), \text{not } h_1, \ldots, \text{not } h_{i-1}, \Gamma\}$$

(The intuition behind the second rule is that we must choose to add h_i if no better literal h_j, $j < i$ is already present in the model.)

% The satisfaction translation, $S(r)$

$$S(r) = \{s(r,1) \leftarrow \text{not } c(r,1), \ldots, \text{not } c(r,k)\} \cup \{s(r,i) \leftarrow c(r,i) \mid 1 \leq i \leq k\}$$

% The options generator, $G(r)$

$$G(r) = \{1\{c(r,1), \ldots, c(r,k)\}1 \leftarrow \Gamma\} \cup \{G(r,i) \mid 1 \leq i \leq k\} \cup S(r)$$

% The generator program, $G(P)$

$$G(P) = \{G(r) \mid r \in P\}$$

Generator Program (cont.)

Example 5 Recall program P_1. $G(P_1)$ is:

$$1\{c(1,1), c(1,2)\}1 \leftarrow \text{not } c.$$
$$a \leftarrow c(1,1), \text{not } c.$$
$$\leftarrow a, \text{not } c(1,1), \text{not } c.$$
$$b \leftarrow c(1,2), \text{not } a, \text{not } c.$$
$$\leftarrow b, \text{not } c(1,2), \text{not } a, \text{not } c.$$

$$s(1,1) \leftarrow \text{not } c(1,1), \text{not } c(1,2).$$
$$s(1,1) \leftarrow c(1,1).$$
$$s(1,2) \leftarrow c(1,2).$$

$$1\{c(2,1), c(2,2)\}1 \leftarrow \text{not } d.$$
$$b \leftarrow c(2,1), \text{not } d.$$
$$\leftarrow b, \text{not } c(2,1), \text{not } d.$$
$$c \leftarrow c(2,2), \text{not } b, \text{not } d.$$
$$\leftarrow c, \text{not } c(2,2), \text{not } b, \text{not } d.$$

$$s(2,1) \leftarrow \text{not } c(2,1), \text{not } c(2,2).$$
$$s(2,1) \leftarrow c(2,1).$$
$$s(2,2) \leftarrow c(2,2).$$

Generator Program (cont.)

Proposition 1 Let P be a LPOD. M is an answer set of $G(P)$ iff $M \cap Lit(P)$ is an answer set of P.

Tester Program

Definition 11 Let P be a LPOD. The *core tester* of the answer set M of P, $C(P, M)$, is:

$$
\begin{aligned}
&C(P, M) = \\
&\quad G(P) \cup \{o(r, i) \mid s(r, i) \in M\} \cup \\
&\quad \{rule(r) \leftarrow \ \mid r \in P\} \cup \\
&\quad \{degree(d) \leftarrow \ \mid \exists\ r \in P \text{ s.t. } r \text{ has at least } d \text{ options}\} \cup \\
&\quad \{\leftarrow \text{not } better\} \cup \\
&\quad \{\leftarrow worse\}
\end{aligned}
$$

The atoms $o(r, i)$ are used to store the degrees of satisfaction in the original answer set M.

Tester Program (cont.)

The tester program depends on a set of rules that are specific to the type of criterion used. Each set of rules are denoted by T_x, where x is either c, i, or p.

The set of rules for Pareto-preference, T_p, is:

$$\begin{aligned} better \leftarrow\ & s(R,I), o(R,J), I < J, rule(R), \\ & degree(I), degree(J). \\ worse \leftarrow\ & s(R,J), o(R,I), I < J, rule(R), \\ & degree(I), degree(J). \end{aligned}$$

Definition 12 The *x-preference tester* ($x \in \{c, i, p\}$) of the answer set M of P, $T_x(P, M)$ is:

$$T_x(P, M) = C(P, M) \cup T_x.$$

Proposition 2 Let P be a LPOD and M an answer set of $G(P)$. M' is an answer set of $T_x(P, M)$ iff $M' \cap Lit(P)$ is an answer set of P which is x-preferred to M.

Corollary 1 Let P be a LPOD and M be an answer set of $G(P)$. M is x-preferred iff $T_x(P, M)$ is inconsistent.

Talk Outline

- Syntax and Answer Sets of Logic Programs with Ordered Disjunction
- Preferred Answer Sets
- Implementation

$\Rightarrow$ **An Application and Conclusions**

Configuration Management

LPOD can be used to model several kinds of preference criteria in software installation.

Typically, different versions of the same software are available. In most cases, we want to install the latest version, but sometimes we have to use an older one (for example, if our computer is too slow to run the most recent version).

The following rule models the desired behavior for the selection of the version of emacs.

$$emacs(21.1) \times emacs(20.7.2) \times emacs(19.34) \leftarrow need(emacs).$$

A more complex example is:

$$libc6 \times libc6\!-\!dev \leftarrow need(libc6), \text{not } c\!-\!developer.$$
$$libc6\!-\!dev \times libc6 \leftarrow need(libc6), c\!-\!developer.$$

Conclusions

Summing up, the important points in this paper are:

- introduction of LPOD with various preference criteria;
- implementation of an inference engine for LPOD;
- in the implementation, use of smodels to determine a “better” answer set;
- specification of preferences between rules *(not shown here)*;
- results on the complexity of LPOD *(not shown here)*.