The PLP System

Toshiko Wakaki[1], Katsumi Inoue[2], Chiaki Sakama[3], and Katsumi Nitta[4]

[1] Shibaura Institute of Technology, twakaki@sic.shibaura-it.ac.jp
[2] National Institute of Informatics, ki@nii.ac.jp
[3] Wakayama University, sakama@sys.wakayama-u.ac.jp
[4] Tokyo Institute of Technology, nitta@dis.titech.ac.jp

1 Introduction

Prioritized Logic Programs (PLPs) [7] introduce explicit representation of priorities to logic programs. They realize various types of (prioritized) commonsense reasoning in artificial intelligence including *preference abduction* [5]. Recently, the authors realize a sound and complete procedure for computing *preferred answer sets* of a PLP [8]. The procedure uses techniques of *answer set programming* (ASP) [6], and also extends the original PLP by accommodating *dynamic preferences* in the language. The PLP procedure is implemented on top of the ASP solver `DLV` [3] using C++, and is now running under the Linux/Windows operating systems as "the PLP system". The system is available at the URL: `http://www.ailab.se.shibaura-it.ac.jp/comppas.html`, which provides binaries of the current release and the instruction on how to use the system. Some examples are also found there.

This paper overviews the system. In Section 2 we review the framework of PLPs and Section 3 presents the algorithm on which the system is based. In Section 4 we solve Gordon's perfected shipping problem in the system, and address brief comparison with related work.

2 Prioritized Logic Programs

A PLP is defined as a pair (P, Φ) where P is a program and Φ is a set of priorities. A program P is a *general extended disjunctive program* (GEDP) [4] which is a set of rules of the form:[5] $L_1; \cdots; L_k; not\ L_{k+1}; \cdots; not\ L_l \leftarrow L_{l+1}, \ldots, L_m,\ not\ L_{m+1}, \ldots, not\ L_n$ $(n \geq m \geq l \geq k \geq 0)$, where each L_i is a positive/negative literal, ";" represents disjunction, and *not* represents negation as failure. A program is *consistent* if it has a *consistent answer set*; otherwise a program is inconsistent. Let Lit be the set of all ground literals in the language of a program, and $\mathcal{L} = Lit \cup \{\, not\ L \mid L \in Lit \,\}$. The pre-order relation $\preceq$ is defined over $\mathcal{L}$. For any $e_1, e_2 \in \mathcal{L}$, $e_1 \preceq e_2$ (called a *priority*) means that e_2 *has a higher priority than* e_1. We write $e_1 \prec e_2$ if $e_1 \preceq e_2$ and $e_2 \not\preceq e_1$. Given a set Φ of priorities, we assume that Φ implicitly involves priorities which are reflexively and transitively derived by those in Φ.

[5] In the system, negation as failure in the head is not used in a program due to the syntactical restriction of `DLV`.

The semantics of a PLP (P, Φ) is given by *preferred answer sets* defined as follows. Let S_1 and S_2 be two answer sets of P. Then, S_2 is *preferable* to S_1 with respect to Φ (written as $S_1 \sqsubseteq S_2$) if for some element $e_2 \in S_2 \setminus S_1$, (i) there is an element $e_1 \in S_1 \setminus S_2$ such that $e_1 \preceq e_2$ is in Φ; and (ii) there is no element $e_3 \in S_1 \setminus S_2$ such that $e_2 \prec e_3$ is in Φ. The relation $\sqsubseteq$ is defined as reflexive and transitive. An answer set S of P is called a *preferred answer set* of a PLP (P, Φ) if $S \sqsubseteq S'$ implies $S' \sqsubseteq S$ for any answer set S' of P. A preferred answer set S is called *strict* if $S \not\sqsubseteq S'$ for any preferred answer set S'. By the definition, preferred answer sets of (P, Φ) are answer sets of P. In particular, preferred answer sets coincide with answer sets when $\Phi = \emptyset$.

3 Computing Preferred Answer Sets

To compute preferred answer sets, we first construct a logic program $T[P, \Phi, S]$ which is obtained from a PLP (P, Φ) and any answer set S of P. The transformation is modular and is done in polynomial-time. We refer the reader to [8] for details of the transformation. The program $T[P, \Phi, S]$ has the following properties.

Theorem 3.1. [8] *Let $T[P, \Phi, S]$ be a program obtained from (P, Φ) and S.*

1. *If $T[P, \Phi, S]$ has a consistent answer set E, then $S' = E \cap Lit$ is an answer set of P satisfying $S \sqsubseteq S'$. Conversely, if there is an answer set S' of P such that $S \sqsubseteq S'$, then $T[P, \Phi, S]$ is consistent.*
2. *$T[P, \Phi, S]$ is inconsistent iff S is a strictly preferred answer set of (P, Φ).*

Given the set $\mathcal{AS}$ of all answer sets of P, we define a bijection $id : \mathcal{AS} \to \Omega$, which maps every answer set $S \in \mathcal{AS}$ to a constant $s \in \Omega$. s is called the *answer set ID* of S. The procedure *CompPAS* computes preferred answer sets of a PLP based on a *generate-and-test* mechanism. Given a PLP (P, Φ), the procedure first computes all answer sets of P using `DLV`. Second, for each answer set S of P, the procedure checks whether S is preferred or not, using $T[P, \Phi, S]$ and the result of Theorem 3.1. Finally it outputs every preferred answer set of the input PLP. The procedure is illustrated in Figure 1.

4 Example: Gordon's Perfected Shipping Problem

In this section we show how the PLP system can solve *Gordon's Perfected Shipping Problem*. The problem is described as follows [1]: A person wants to find out if her security interest in a certain ship is perfected. According to the Uniform Commercial Code (UCC), a security interest in goods may be perfected by taking possession of the collateral. On the other hand, according to a federal law called the Ship Mortgage Act (SMA), a security interest in a ship may only be perfected by filing a financing statement. She currently has possession of the ship, but a statement has not been filed. Now a question is whether the UCC or the SMA takes precedence in this case.

The situation is represented by the program:

Procedure: $CompPAS(P, \Phi, \Delta)$

Input : a PLP (P, Φ);
Output : the set Δ of all preferred answer sets of (P, Φ).

1. Compute the set $\mathcal{AS}$ of all answer sets of P.
2. If $\Phi = \emptyset$, return $\Delta = \mathcal{AS}$; otherwise, do:
   (a) Set $\Sigma := \emptyset$.
   (b) For each $S \in \mathcal{AS}$, if $T[P, \Phi, S]$ is consistent, do the following steps:
      i. for each answer set E of $T[P, \Phi, S]$, put $S' = E \cap Lit$ and find the answer set ID $s' \in \Omega$ for S' where $S' \in \mathcal{AS}$ by Theorem 3.1;
      ii. put $\Sigma := \Sigma \cup \{ \sqsubseteq (s, s') \leftarrow \}$.
3. Compute an answer set U of the program: $\Psi \cup \Sigma \cup \{ as(s) \leftarrow \mid \ s \in \Omega \}$ where Ψ consists of the following five rules:

   $\sqsubseteq (x, x) \leftarrow as(x), \quad \sqsubseteq (x, z) \leftarrow \sqsubseteq (x, y), \sqsubseteq (y, z), \quad \sqsubset (x, y) \leftarrow \sqsubseteq (x, y), \mathit{not} \sqsubseteq (y, x),$
   $worse(x) \leftarrow \sqsubset (x, y), \quad p\text{-}as(x) \leftarrow as(x), \mathit{not}\ worse(x).$

4. Return $\Delta = \{ S \mid S \in \mathcal{AS} \text{ s.t. } s = id(S) \text{ and } p\text{-}as(s) \in U \}$.

Fig. 1. The procedure CompPAS

```
perfected :- poss, not ab1.
-perfected :- ship, -file, not ab2.
poss.    ship.    -file.
ab1 :- not ab2.    ab2 :- not ab1.
ucc :- not ab1.   sma :- not ab2.
```

Two answer sets: $S_1 = \{ \texttt{poss}, \texttt{ship}, -\texttt{file}, \texttt{ab1}, -\texttt{perfected}, \texttt{sma} \}$ and $S_2 = \{ \texttt{poss}, \texttt{ship}, -\texttt{file}, \texttt{ab2}, \texttt{perfected}, \texttt{ucc} \}$ are computed by `DLV`, which represents two conflicting solutions.

To resolve conflict, the principle of *Lex Posterior* gives precedence to newer laws. In this example, the UCC is newer than the SMA. On the other hand, the principle of *Lex Superior* gives precedence to laws supported by the higher authority. The SMA is a federal law and has higher authority here. These priority knowledge is encoded in Φ as *priorities with preconditions* as follows:

```
moreRecent(ucc,sma).   fed(sma).  state(ucc).
lp(Y,X) :- moreRecent(X,Y).    ls(Y,X) :- fed(X), state(Y).
preceq(Y,X) :- lp(Y,X), not conf1(X,Y).
preceq(Y,X) :- ls(Y,X), not conf1(X,Y).
```

Here `preceq(Y,X)` means $Y \preceq X$ and `conf1` is a predicate for resolving conflict between legal principles. The last two rules express *dynamic preferences* which enable us to specify context-dependent preferences [1]. With these additional rules, S_1 and S_2 become two *tie-preferred* answer sets, i.e., $S_1 \sqsubseteq S_2$ and $S_2 \sqsubseteq S_1$

Finally, we add additional priority information to Φ that states *Lex Superior* has precedence to *Lex Posterior*:

```
conf1(Y,X) :- lp(X,Y), ls(Y,X), not conf2(X,Y),
```

where `conf2` is a predicate for resolving conflict which is one level higher than `conf1`. With this rule `conf1(ucc, sma)` is derived, which blocks the derivation of `preceq(sma, ucc)`. As a result, `preceq(ucc, sma)` is derived, and we get the single preferred answer set S_1 as the solution.

The above PLP program `gordon` is set as an input to the system, then `compplp` computes the preferred answer sets as follows:

```
$ compplp -timec gordon
Total Number of Answer Sets of P:2
{-file, -perfected, ab1, poss, ship, sma}
{-file, ab2, perfected, poss, ship, ucc}

Total Number of Preferred Answer Sets of (P,Phi):1
{-file, -perfected, ab1, poss, ship, sma}
Time: 10msec
```

We finally address brief comparison with related work. Brewka *et al*. [2] develop *logic programs with ordered disjunction* (LPOD) using an answer set solver. Their implementation is based on two logic programs, a generator and a tester to compute preferred answer sets. According to [2], an LPOD with the *Pareto-preference* is effectively encoded in a PLP, but the converse direction, computing preferred answer sets of a PLP in terms of an LPOD, is unlikely possible in general. The difficulty of reverse encoding is justified from the complexity viewpoints: reasoning tasks in LPODs lie at the second level of the polynomial hierarchy, while those in PLPs lie at the third level. Reasoning tasks in PLPs are thus generally hard, but as shown in the above example, the system performance is encouraging in many real world problems.

Acknowledgments: The authors thank Yosuke Kiwada at Shibaura Institute of Technology for his assistance of implementing the system.

References

1. G. Brewka. Well-founded semantics for extended logic programs with dynamic preferences. *J. Artificial Intelligence Research* 4, pp. 19–36, 1996.
2. G. Brewka, I. Niemelä, and T. Syrjänen. Logic programs with ordered disjunction. *Computational Intelligence* 20, pp. 335-357, 2004.
3. T. Eiter, W. Faber, N. Leone, and G. Pfeifer. Declarative problem solving using the DLV system. *Logic-Based Artificial Intelligence*, Kluwer, pp. 79–103, 2000.
4. K. Inoue and C. Sakama. Negation as failure in the head. *J. Logic Programming* 35(1), pp. 39–78, 1998.
5. K. Inoue and C. Sakama. Abducing priorities to derive intended conclusions. *Proc. IJCAI-99*, Morgan Kaufmann, pp. 44–49.
6. V. Lifschitz. Answer set programming and plan generation. *Artificial Intelligence* 138, pp. 39–54, 2002.
7. C. Sakama and K. Inoue. Prioritized logic programming and its application to commonsense reasoning. *Artificial Intelligence* 123, pp. 185–222, 2000.
8. T. Wakaki, K. Inoue, C. Sakama, and K. Nitta. Computing preferred answer sets in answer set programming. *Proc. 10th Int'l Conf. Logic for Programming, Artificial Intelligence, and Reasoning (LPAR'03)*, LNAI 2850, Springer, pp. 259–273, 2003.