# plp: A Generic Compiler for Ordered Logic Programs

James P. Delgrande[1], Torsten Schaub[2⋆], and Hans Tompits[3]

[1] School of Computing Science, Simon Fraser University, Burnaby, B.C., Canada V5A 1S6,
jim@cs.sfu.ca
[2] Institut für Informatik, Universität Potsdam, Postfach 60 15 53, D–14415 Potsdam, Germany,
torsten@cs.uni-potsdam.de
[3] Institut für Informationssysteme, Abt. Wissensbasierte Systeme 184/3,
Technische Universität Wien, Favoritenstraße 9–11, A–1040 Vienna, Austria,
tompits@kr.tuwien.ac.at

**Abstract** This paper describes a generic compiler, called plp, for translating ordered logic programs into standard logic programs under the answer set semantics. In an ordered logic program, preference information is expressed at the object level by atoms of the form $s \prec t$, where $s$ and $t$ are names of rules. An ordered logic program is transformed into a second, regular, extended logic program wherein the preferences are respected, in that the answer sets obtained in the transformed theory correspond with the preferred answer sets of the original theory. Currently, plp treats three different types of preference strategies, viz. those proposed by (i) Brewka and Eiter, (ii) Delgrande, Schaub, and Tompits, and (iii) Wang, Zhou, and Lin. Since the result of the translation is an extended logic program, existing logic programming systems can be used as underlying reasoning engine. In particular, plp is conceived as a front-end to the logic programming systems dlv and smodels.

## 1 General information

Several approaches have been introduced in recent years for expressing preference information within declarative knowledge representation formalisms [7,11,1,10]. However, most of these methods treat preferences at the meta-level and require a change of the underlying semantics. As a result, implementations need in general fresh algorithms and cannot rely on existing systems computing the regular (unordered) formalisms.

In this paper, we describe the system plp, which avoids the need of new algorithms, while computing preferred answer sets of an ordered logic program. plp is based on an approach for expressing preference information *within* the framework of standard answer set semantics [6], and is conceived as a front-end to the logic programming systems dlv [5] and smodels [8]. The general technique is described in [4] and derives from a methodology for addressing preferences in default logic first proposed in [2].

We begin with an *ordered logic program*, which is an extended logic program in which rules are named by unique terms and in which preferences among rules are given by a new set of atoms of the form $s \prec t$, where $s$ and $t$ are names. Such an ordered logic program is then transformed into a second, regular, extended logic program wherein the

⋆ Affiliated with the School of Computing Science at Simon Fraser University, Burnaby, Canada.

| Meaning | Symbols | Internal |
|---|---|---|
| $\bot, \top$ | `false/0`, `true/0` | |
| $\neg$ | `neg/1`, `-/1` (prefix) | `neg_`$L$, $L \in \mathcal{L}$ |
| $not$ | `not/1`, $\sim$`/1` (prefix) | |
| $\wedge$ | `,/1` (infix; in body) | |
| $\vee$ | `;/1`, `v/2`, `\|/2` (infix; in head) | |
| $\leftarrow$ | `:-/1` (infix; in rule) | |
| $\prec$ | `</2` (infix) | `prec/2` |
| $n_r : \langle head(r) \rangle \leftarrow \langle body(r) \rangle$ | $\langle head(r) \rangle$ `:-` `name`$(n_r), \langle body(r) \rangle$<br>$\langle head(r) \rangle$ `:-` $[n_r], \langle body(r) \rangle$ | |
| ok, rdy | | `ok/1`, `rdy/2` |
| ap, bl | | `ap/1`, `bl/1` |

**Table 1.** The syntax of plp input files.

preferences are respected, in the sense that the answer sets obtained in the transformed theory correspond to the preferred answer sets of the original theory. The transformation is realized by adding sufficient control elements to the rules of the given ordered logic program which guarantee that successive rule applications are in accord with the intended order. More specifically, the transformed program contains control atoms ap$(\cdot)$ and bl$(\cdot)$, which detect when a rule has been applied or blocked, respectively, as well as auxiliary atoms ok$(\cdot)$ and rdy$(\cdot,\cdot)$ which control the applicability of rules based on antecedent conditions reflecting the given order information.

The approach is sufficiently general to allow the specification of preferences among preferences, preferences holding in a particular context, and preferences holding by default. Moreover, the approach permits a generic compilation methodology, making it possible to express differing preference strategies. Basically, this is achieved by varying the specific antecedent conditions for the control atoms ok$(\cdot)$ and rdy$(\cdot,\cdot)$. Currently, plp treats three kinds of preference strategies, viz. those proposed by Brewka and Eiter [1], Delgrande, Schaub, and Tompits [2,4], and Wang, Zhou, and Lin [10].

## 2 Applying the system

The syntax of plp is summarised in Table 1. An example file comprising an ordered logic program is the following:

```
    neg a .
        b :- name(n2), neg a, not c.
        c :- name(n3), not b.
(n3 < n2) :- not d.
```

Here, `name(n2)` and `name(n3)` serve as names for the rules in which these terms occur, and the last rule expresses that the rule named `n2` is preferred over the rule named `n3`, in case atom `d` cannot be inferred.

Once this file, say `example.lp`, is read into plp, it is subject to multiple transformations. Most of these transformations are rule-centered in the sense that they apply in

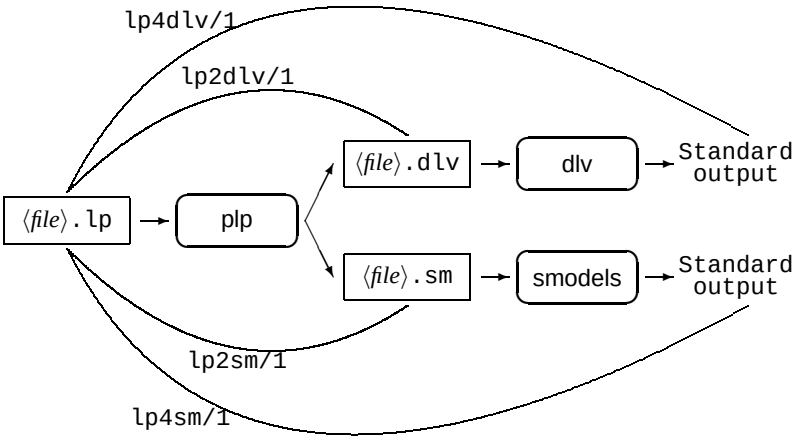


**Figure 1.** Compilation with plp: external view.

turn to each single rule. The first phase of the compilation is system-independent and corresponds to the transformations given in [4]. While the original file is supposed to have the extension `lp`, the result of the system-independent compilation phase is kept in an intermediate file with extension `pl` (e.g., `example.pl`).

While this compilation phase can be engaged explicitly by the command `lp2pl/1`, one is usually interested in producing system-specific code that is directly usable by either dlv or smodels. This can be done by means of the commands `lp2dlv/1` and `lp2sm/1`,[1] which then produce system-specific code resulting in files having extensions `dlv` and `sm`, respectively. These files can then be fed into the respective system by a standard command interpreter, such as a UNIX shell, or from within the Prolog system through commands `dlv/1` or `smodels/1`. For example, after compiling our example by `lp2dlv`, we may proceed as follows:

```
| ?- dlv('Examples/example').
Calling :dlv  Examples/example.dlv
dlv [build BEN/Jun 11 2001   gcc 2.95.2 19991024 (release)]

{true, name(n2), name(n3), neg_a, ok(n2), rdy(n2,n2),
 rdy(n2,n3), rdy(n3,n3), prec(n3,n2), neg_prec(n2,n3),
 ap(n2), b, rdy(n3,n2), ok(n3), bl(n3)}
```

Both commands can be furnished with the option `nice` (as an additional argument) in order to strip off the auxiliary predicates:

```
| ?- dlv('Examples/example',nice).
Calling :dlv  -filter=a [...] -filter=neg_d Examples/example.dlv
dlv [build BEN/Jun 11 2001   gcc 2.95.2 19991024 (release)]

{neg_a, b}
```

[1] These files are themselves obtainable from the intermediate `pl`-files via commands `pl2dlv/1` and `pl2sm/1`, respectively.

The above series of commands can be engaged within a single one by means of `lp4dlv/1` and `lp4sm/1`, respectively. Moreover, for changing the underlying preference strategy, a simple patch is executed, which redefines certain predicates. The overall (external) comportment of `plp` is illustrated in Figure 1.

For treating variables, some additional preprocessing is necessary for instantiating the rules before they are compiled. The presence of variables is indicated by file extension `vlp`. The content of such a file is first instantiated by systematically replacing variables by constants and then freed from function symbols by replacing terms by constants, e.g., `f(a)` is replaced by `f_a`. This is clearly a rather pragmatic approach. A more elaborated compilation would be obtained by proceeding right from the start in a system-specific way.

Finally, the current prototype is available at
`http://www.cs.uni-potsdam.de/~torsten/plp/`.

*Acknowledgements.* The first author received partial support from the Natural Sciences and Engineering Research Council of Canada; the second author was partially supported by the German Science Foundation (DFG) under grant FOR 375/1-1, TP C; the third author was partially supported by the Austrian Science Fund (FWF) under grant P13871-INF.

## References

1. G. Brewka and T. Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356, 1999.
2. J. Delgrande and T. Schaub. Compiling reasoning with and about preferences into default logic. In *Proc. IJCAI-97*, pages 168–174. Morgan Kaufmann Publishers, 1997.
3. J. Delgrande, T. Schaub, and H. Tompits. A compilation of Brewka and Eiter's approach to prioritization. In *Proc. JELIA-00*, pages 376–390. Springer Verlag, 2000.
4. J. Delgrande, T. Schaub, and H. Tompits. Logic programs with compiled preferences. In *Proc. ECAI-00*, pages 392–398. IOS Press, 2000.
5. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for nonmonotonic reasoning. In *Proc. LPNMR-97*, pages 363–374. Springer Verlag, 1997.
6. M. Gelfond and V. Lifschitz. Classical negation in logic programs and deductive databases. *New Generation Computing*, 9:365–385, 1991.
7. M. Gelfond and T. Son. Reasoning with prioritized defaults. In *Third International Workshop on Logic Programming and Knowledge Representation*, pages 164–223. Springer Verlag, 1997.
8. I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal logic programs. In *Proc. LPNMR-97*, pages 420–429. Springer Verlag, 1997.
9. T. Schaub and K. Wang. A comparative study of logic programs with preference: Preliminary report. In *Proc. AAAI Spring Symposium on Answer Set Programming*, pages 151–157. AAAI Press, 2001.
10. K. Wang, L. Zhou, and F. Lin. Alternating fixpoint theory for logic programs with priority. In *Proc. First International Conference on Computational Logic*, pages 164–178. Springer Verlag, 2000.
11. Y. Zhang and N. Foo. Answer sets for prioritized logic programs. In *Proc. ILPS-97*, pages 69–84. MIT Press, 1997.