

# Logic Programming with Ordered Disjunction

Gerhard Brewka

Universität Leipzig  
Institut für Informatik  
Augustusplatz 10-11  
04109 Leipzig, Germany  
brewka@informatik.uni-leipzig.de

## Abstract

Logic programs with ordered disjunction (*LPODs*) combine ideas underlying Qualitative Choice Logic (Brewka, Benferhat, & Le Berre 2002) and answer set programming. Logic programming under answer set semantics is extended with a new connective called ordered disjunction. The new connective allows us to represent alternative, ranked options for problem solutions in the heads of rules:  $A \times B$  intuitively means: if possible  $A$ , but if  $A$  is not possible then at least  $B$ . The semantics of logic programs with ordered disjunction is based on a preference relation on answer sets. *LPODs* are useful for applications in design and configuration and can serve as a basis for qualitative decision making.

## Introduction

In a recent paper (Brewka, Benferhat, & Le Berre 2002) a propositional logic called Qualitative Choice Logic (*QCL*) was introduced. The logic contains a new connective  $\times$  representing ordered disjunction. Intuitively,  $A \times B$  stands for: if possible  $A$ , but if  $A$  is impossible then (at least)  $B$ . This connective allows context dependent preferences to be represented in a simple and elegant fashion. As a simple example consider the preferences for booking a hotel for a conference. Assume the most preferred option is to be within walking distance from the conference site, the second best option is to have transportation provided by the hotel, the third best is public transportation. This can simply be represented as

$walking \times hotel\text{-}transport \times public\text{-}transport$

From a description of available hotels, a disjunction expressing that one of the hotels must be picked, and the above formula *QCL* is able to derive the hotel which satisfies best the given preferences (if there is more than one such hotel a corresponding disjunction is concluded).

The semantics of the logic is based on degrees of satisfaction of a formula in a classical model. The degrees, intuitively, measure disappointment and induce a preference relation on models. Consequence is defined in terms of most preferred models. It is argued in that paper that there are numerous useful applications, e.g. in configuration and design.

Copyright © 2002, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

In this paper we want to combine ideas underlying *QCL* with logic programming. More precisely, we want to investigate logic programs based on rules with ordered disjunction in the heads. We call such programs logic programs with ordered disjunction (*LPODs*).

The semantical framework in which the investigation will be carried out is that of answer set semantics (Gelfond & Lifschitz 1991). Logic programs under answer set semantics have emerged as a new promising programming paradigm dubbed answer set programming. There are numerous interesting AI applications of answer set programming, for instance in planning (Lifschitz 2001) and configuration (Soininen 2000). One of the reasons for this success is the availability of highly efficient systems for computing answer sets like *smodels* (Niemelä & Simons 1997) and *dlv* (Eiter *et al.* 1998).

We think it is worthwhile to investigate simple representations of context dependent preferences in the answer set programming paradigm. Our combination of ideas from *QCL* and answer set programming will lead to an approach which is less expressive than *QCL* in one respect: the syntax of *LPODs* restricts the appearance of ordered disjunction to the head of rules. On the other hand, we inherit from answer set programming the nonmonotonic aspects which are due to default negation. This allows us to combine default knowledge with knowledge about preferences and desires in a simple and elegant way.

The basic intuition underlying our approach can be described as follows: we will use the ordered disjunctions in rule heads to select some of the answer sets of a program as the preferred ones. Consider a program containing the rule

$$A \times B \leftarrow C$$

If  $S_1$  is an answer set containing  $C$  and  $A$  and  $S_2$  is an answer set containing  $C$  and  $B$  but not  $A$ , then - *ceteris paribus* (other things being equal) -  $S_1$  is preferred over  $S_2$ . Of course, we have to give precise meaning to the *ceteris paribus* phrase. Intuitively *ceteris paribus* is to be read as  $S_1$  and  $S_2$  satisfy the other rules in the program equally well.

We will show that under certain conditions reasoning from most preferred answer sets yields optimal problem solutions. In more general decision making settings the preference relation on answer sets provides a basis for best possible choices given a specific decision strategy.

We will restrict our discussion in this paper to propositional programs. However, as usual in answer set programming, we admit rule schemata containing variables bearing in mind that these schemata are just convenient representations for the set of their ground instances.

The rest of the paper is organized as follows. In the next section we introduce syntax and semantics of *LPODs*. We define the degree of satisfaction of a rule in an answer set and show how to use the degrees to determine a preference relation on answer sets. Conclusions are defined as the literals true in all preferred answer sets. The subsequent section discusses some simple examples and potential applications. We then investigate implementation issues. The last section discusses related work and concludes.

## Logic programs with ordered disjunction

Logic programming with ordered disjunction is an extension of logic programming with two kinds of negation (default and strong negation) (Gelfond & Lifschitz 1991). The new connective  $\times$  representing ordered disjunction is allowed to appear in the head of rules only. A (propositional) *LPOD* thus consists of rules of the form

$$C_1 \times \dots \times C_n \leftarrow A_1, \dots, A_m, \text{not } B_1, \dots, \text{not } B_k$$

where the  $C_i$ ,  $A_j$  and  $B_l$  are ground literals.

The intuitive reading of the rule head is: if possible  $C_1$ , if  $C_1$  is not possible then  $C_2$ , ..., if all of  $C_1, \dots, C_{n-1}$  are not possible then  $C_n$ . The literals  $C_i$  are called choices of the rule. Extended logic programs with two negations are a special case where  $n = 1$  for all rules. As usual we omit  $\leftarrow$  whenever  $m = 0$  and  $k = 0$ , that is, if the rule is a fact. Moreover, rules of the form  $\leftarrow \text{body}$  (constraints) are used as abbreviations for  $p \leftarrow \text{body}$ ,  $\text{not } p$  for some  $p$  not appearing in the rest of the program. The effect is that no answer sets containing *body* exist.

Before defining the semantics of *LPODs* a few observations are in order. As already mentioned in the introduction we want to use the ranking of literals in the head of rules to select some of the answer sets of a program as the preferred ones. But what are the answer sets of a program among which to make this selection?

Since ordered disjunction is a particular prioritized form of disjunction it seems like a natural idea to base the semantics of *LPODs* on one of the standard semantics for disjunctive logic programs, for instance Gelfond and Lifschitz's semantics (Gelfond & Lifschitz 1991).

Unfortunately, this doesn't work. The problem is that most of the semantics for disjunctive logic programs have minimality built in. For instance, according to Gelfond and Lifschitz,  $S$  is an answer set of a disjunctive logic program  $P$  iff  $S$  is a minimal set of literals which is logically closed, and closed under the  $S$ -reduct of  $P$ . The  $S$ -reduct of  $P$  is obtained from  $P$  by (1) deleting all rules  $r$  from  $P$  such that  $\text{not } B_j$  in the body of  $r$  and  $B_j \in S$ , and (2) deleting all default negated literals from the remaining rules. A set of literals  $S$  is closed under a rule  $r$  if one of the literals in the head of  $r$  is in  $S$  whenever the body is true in  $S$  (see (Gelfond & Lifschitz 1991) for the details).

In this approach answer sets are minimal: if  $S_1$  and  $S_2$  are answer sets of a disjunctive program  $P$  and  $S_1 \subseteq S_2$ , then  $S_2 \subseteq S_1$ .

Minimality is not always wanted for *LPODs*. Consider the following two facts:

- 1)  $A \times B \times C$
- 2)  $B \times D$

The single best way of satisfying both ordered disjunctions is obviously to make  $A$  and  $B$  true, that is, we would expect  $\{A, B\}$  to be the single preferred answer set of this simple *LPOD*. However, since  $B$  is sufficient to satisfy both disjunctions, the set  $\{A, B\}$  is not even an answer set of the corresponding disjunctive logic program (where  $\times$  is replaced by  $\vee$ ) according to the semantics of (Gelfond & Lifschitz 1991): the built in minimality precludes sets containing both  $A$  and  $B$  from consideration.

We thus have to use a semantics which is not minimal. Indeed, there is such a semantics, the possible models semantics proposed by Sakama and Inoue (Sakama & Inoue 1994). It is based on so-called split programs, that is, disjunction free programs which contain arbitrary subsets of single head rules obtained from disjunctive rules by deleting all but one alternatives in the head.

Unfortunately, also this semantics is inadequate, this time for opposite reasons: it admits too many literals in answer sets. Consider the disjunctive logic program

- 1)  $A \vee B \vee C$

There are seven split programs corresponding to the nonempty subsets of the literals of the fact. The split program containing the facts  $A, B, C$  generates the possible model where  $A, B, C$  is true.

Let us replace disjunction by ordered disjunction in this formula. According to our intuitive discussion we want to read the rule as "if possible  $A$ , if this is not possible then  $B$ , and if also  $B$  is not possible then  $C$ ". Under this reading models containing more than one of the literals in the head do not seem justified on the basis of a single rule (they may be justified by different rules, though).

For this reason we will not allow cases where a single rule of the original program gives rise to more than one rule in the split program. There is a further complication: consider the program:

- 1)  $A \times B \times C$
- 2)  $A$

We do not want to obtain  $\{A, B\}$  as an answer set from the split program consisting of these 2 atomic facts since again this does not correspond to the intuitive reading of the first rule ( $B$  only if  $A$  is not possible). We therefore have to use slightly more complicated rules in split programs.

**Definition 1** Let  $r = C_1 \times \dots \times C_n \leftarrow \text{body}$  be a rule. For  $k \leq n$  we define the  $k$ th option of  $r$  as

$$r^k = C_k \leftarrow \text{body}, \text{not } C_1, \dots, \text{not } C_{k-1}.$$

**Definition 2** Let  $P$  be an *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.



Next assume that, in addition to *summer* also the literal  $\neg$ hot is given. The single preferred answer set now is  $S_3 = \{summer, \neg hot, cinema\}$ . All rules are satisfied to degree 1.

Finally, assume the additional facts are *summer* and *rain*. Now the single preferred answer set (and in fact the single answer set) is

$$S_4 = \{summer, rain, hot, \neg beach, cinema\}.$$

Note that this time it is not possible to satisfy all rules to degree 1: rule 2) is satisfied to degree 2 only. As often in real life, there are situations where the best options simply do not work out.

We think that *LPODs* are well suited for representing problems where choices under preferences have to be made, for instance in cases where a number of components have to be chosen for a certain configuration task. The general idea would be to have

- for each component a set of rules describing its properties,
- rules describing which components are needed for the configuration to be complete; this may depend on other components chosen,
- rules describing intended properties of the solution we want to generate. The involved preferences may be context dependent, and
- a description of the case at hand.

In each case default knowledge can be used to describe what is normally the case. Consider the problem of configuring a menu. The menu should consist of a *starter*, a *main course*, a *dessert* and a *beverage*. As a *starter* you prefer *soup* over *salad*. As *main course* *fish*, *beef* and *lasagne* are possible (this is all you are able to cook) and your preferences are in this order. Of course, if the visitor is *vegetarian* the first two (as well as the *soup*) are out of the question. In case of *beef* you prefer *red* wine over *white* wine over mineral *water*, otherwise the order between wines is reversed. Only *ice-coffee* and *tiramisu* is available as a *dessert*. If *tiramisu* is chosen, then an extra *coffee* is necessary. You prefer *espresso* over *cappucino*.

The possible components thus are *soup*, *salad*, *fish*, *beef*, *lasagne*, *ice-coffee*, *tiramisu*, *espresso*, *cappucino*, *red*, *white* and *water*. The following properties of the components are relevant:

$$\begin{aligned} \neg vegetarian &\leftarrow beef & alcohol &\leftarrow white \\ \neg vegetarian &\leftarrow fish & alcohol &\leftarrow red \\ \neg vegetarian &\leftarrow soup \end{aligned}$$

The needed components are

$$\begin{aligned} starter & & beverage \\ main & & coffee \leftarrow tiramisu \\ dessert \end{aligned}$$

The preferences are as follows:

$$\begin{aligned} soup \times salad &\leftarrow starter \\ fish \times beef \times lasagne &\leftarrow main \\ red \times white \times water &\leftarrow beverage, beef \\ white \times red \times water &\leftarrow beverage, not beef \end{aligned}$$

$$\begin{aligned} espresso \times cappuccino &\leftarrow coffee \\ ice-coffee &\leftarrow not tiramisu, dessert \\ tiramisu &\leftarrow not ice-coffee, dessert \end{aligned}$$

Now, given a description of the case at hand, e.g. whether the visitor is vegetarian or not, drinks alcohol or not, likes fish etc. the preferred answer sets will determine a menu which satisfies the preferences as much as possible. The last two rules are necessary to make sure that one of the desserts is picked. For the other courses this is implicit in the specified preferences. In the language of (Niemelä & Simons 2000) these rules can be represented as the cardinality constraint rule  $1\{ice-coffee, tiramisu\}1 \leftarrow dessert$ . Combinations of *LPODs* and such constraints are a topic of further research.

## Computation

The first question to ask is whether *LPODs* can simply be reduced to standard logic programs with two kinds of negation. In that case standard answer set programming techniques would be sufficient for computing consequences of *LPODs*. We will show that a seemingly natural translation does not yield the intended answer sets.

**Definition 8** The pseudo-translation  $trans(r)$  of a rule

$$r = C_1 \times \dots \times C_n \leftarrow body$$

is the collection of rules

$$\begin{aligned} C_1 &\leftarrow body, not \overline{C_1} \\ C_2 &\leftarrow body, not \overline{C_2}, \overline{C_1} \\ \dots \\ C_{n-1} &\leftarrow body, not \overline{C_{n-1}}, \overline{C_1}, \dots, \overline{C_{n-2}} \\ C_n &\leftarrow body, \overline{C_1}, \dots, \overline{C_{n-1}} \end{aligned}$$

where  $\overline{C}$  is the complement of  $C$ , that is  $\neg C$  if  $C$  is an atom and  $C'$  if  $C = \neg C'$ . The pseudo-translation  $trans(P)$  of an *LPOD*  $P$  is

$$trans(P) = \bigcup_{r \in P} trans(r)$$

The pseudo-translation creates for each option  $C_i$  in the head of  $r$  a rule with head  $C_i$  which has the negation of the better options as additional body literals. In addition, the rule is made defeasible by adding the default negation of the complement of  $C_i$  to the body. There is an exception: the rule generated for the last option is not made defeasible this way since at least one of the options must be true whenever the body of the original rule is true.

Although this translation seems natural it does not work. Consider the following example:

- 1)  $a \times b$
- 2)  $p \leftarrow not p, a$

The single preferred answer set is  $\{b\}$ . The pseudo-translation is

- 1)  $a \leftarrow not \neg a$
- 2)  $b \leftarrow \neg a$
- 3)  $p \leftarrow not p, a$

The resulting program has no answer set. In fact, we can prove the following proposition:

**Proposition 2** *There is no translation trans from LPODs to extended logic programs (without ordered disjunction) such that for each program P the preferred answer sets of P and the answer sets of trans(P) coincide.*

**Proof:** The proposition follows from the fact that preferred answer sets of LPODs are not necessarily subset minimal. Consider the program  $a \times b; c \times b \leftarrow a; \neg c$ . The preferred answer sets are  $S_1 = \{b, \neg c\}$  and  $S_2 = \{a, b, \neg c\}$ . Clearly,  $S_1 \subset S_2$ . There is thus no extended logic program with these answer sets.  $\square$

Of course, this does not exclude the possibility of translations to programs containing some extra atoms. This is a topic of further study.

For the computation of preferred answer sets we suggest to search through the space of split programs. Fortunately, the split programs can be ordered according to the options they contain. Let us first introduce some notation. For an LPOD P and a split program P' of P we let

$$P_{P'}^k = \{r \in P \mid k \text{ smallest integer such that } r^k \in P'\}.$$

Now consider the following preference relation on split programs:

**Definition 9** *Let  $P_1$  and  $P_2$  be two split programs of the LPOD P. We define  $P_1 > P_2$  iff there is a k such that  $P_{P_2}^k \subset P_{P_1}^k$  and for all  $j < k$ ,  $P_{P_1}^j = P_{P_2}^j$ .*

The split programs of P form a lattice whose top element is the program with all the best options and whose bottom element is the program with all the worst options. The lub of two programs picks for each rule in P the best option contained in one of the programs, the glb picks the worst option.

Among the split programs we are interested in consistent ones which are  $>$ -maximal. A program is consistent if it possesses at least one consistent answer set.

**Definition 10** *A split program P' of P is called P-optimal iff P' is consistent, and there is no consistent split program P'' of P such that  $P'' > P'$ .*

We are able to prove the following proposition:

**Proposition 3** *If S is a preferred answer set of an LPOD P then there is a P-optimal split program P' of P such that S is an answer set of P'.*

**Proof:** Sketch: Let S be a preferred answer set of P. Construct a split program P' as follows: for each rule  $r = C_1 \times \dots \times C_n \leftarrow body \in P$  choose  $r^1$  if body is false in S and  $r^k$  if body is true in S and k is the smallest integer such that  $C_k \in S$ . The split program obtained this way has S as an answer set and can be shown to be optimal.  $\square$

We can thus use the preference ordering on split programs to search for preferred answer sets, starting from the best split program. Whenever a split program possesses a consistent answer set we can eliminate all programs below from further consideration.

## Conclusion

In this paper we introduced a new connective to logic programming. This connective - called ordered disjunction - can be used to represent context dependent preferences in a simple and elegant way. Logic programming with ordered disjunction has interesting applications, in particular in design and configuration.

There are numerous papers introducing preferences to logic programming. For an overview of some of these approaches see the discussion in (Brewka & Eiter 1999) or the more recent (Schaub & Wang 2001). Only few of these proposals allow for context dependent preferences. Such preferences are discussed for instance in (Brewka 1996; Brewka & Eiter 1999). The representation of the preferences in these papers is based on the introduction of names for rules, the explicit representation of the preference relation among rules in the logical language, and a sophisticated reformulation of the central semantic notion (answer set, extension, etc.) with a highly self-referential flavour. Alternative approaches (Delgrande, Schaub, & Tompits 2000; Grosz 1999) are based on compilation techniques and make heavy use of meta-predicates in the logical language. Nothing like this is necessary in our approach. All we have to do is use the degree of satisfaction of a rule to define a preference relation on answer sets directly.

Our approach is closely related to work in qualitative decision theory, for an overview see (Doyle & Thomason 1999). Poole (Poole 1997) aims at a combination of logic and decision theory. His approach incorporates quantitative utilities whereas our preferences are qualitative. Interestingly, Poole uses a logic *without* disjunction whereas we *enhance* disjunction. In (Boutilier *et al.* 1999) a graphical representation, somewhat reminiscent of Bayes nets, for conditional preferences among feature values under the *ceteris paribus* principle is proposed, together with corresponding algorithms. LPODs are more general and offer means to reason defeasibly. Several models of qualitative decision making based on possibility theory are described in (Dubois *et al.* 1999; Benferhat *et al.* 2000). They are based on certainty and desirability rankings. Some of them make strong commensurability assumptions with respect to these rankings. In a series of papers (Lang 1996; van der Torre & Weydert 2001), originally motivated by (Boutilier 1994), the authors propose viewing conditional desires as constraints on utility functions. Intuitively,  $D(a|b)$  stands for: the b-worlds with highest utility satisfy a. Our interpretation of ranked options is different. Rather than being based on decision theory our approach gives a particular interpretation to the *ceteris paribus* principle.

In an extended version of this paper (available at [www.informatik.uni-leipzig.de/~brewka](http://www.informatik.uni-leipzig.de/~brewka)) we show that LPODs can serve as a basis for qualitative decision models. In decision making it is not sufficient to consider the most preferred answer sets only since this amounts to an extremely optimistic view about how the world will behave. As is well-known in decision theory, for realistic models of decision making it is necessary to distinguish what is under the control of the agent (and thus may constitute the agent's decision) from what is not. This can be done by distinguish-

ing a subset of the literals in a program as decision literals. The basic idea is to use *LPODs* to describe possible actions or decisions and their consequences, states of the world and desired outcomes. The necessary steps are:

1. Among the literals in the logical language distinguish a set of decision literals  $C$  the agent can decide upon. It's the agent's decision which makes them true. A decision is a consistent subset of  $C$ .
2. Represent the different alternative decisions which can be made by the agent. This can be done using standard answer set programming techniques. Note that certain options may lead to additional choices that need to be made.
3. Represent the different possible states of the world. Again standard answer set programming techniques apply.
4. Represent relationships between and consequences of different alternatives.
5. Represent desired properties. This is where ordered disjunction comes into play. Of course, desires may be context-dependent.
6. Use the preference relation on answer sets derived from the satisfaction degrees of rules to induce a preference relation on possible decisions. There are different ways to do this corresponding to different attitudes towards risk.
7. Pick one of the most preferred decisions.

We plan to investigate application methodologies for *LPODs* in decision making and other scenarios. An answer set programming methodology for configuration tasks was developed by Niemelä and colleagues at Helsinki University of Technology (Soininen 2000; Niemelä & Simons 2000). We plan to study possibilities of combining this methodology with *LPODs*. Finally, we are studying possibilities of implementing *LPODs* on top of the *smodels* system.

### Acknowledgements

Thanks to S. Benferhat, R. Booth, T. Janhunen, I. Niemelä, T. Syrjänen and L. van der Torre for helpful comments.

### References

- Benferhat, S.; Dubois, D.; Fargier, H.; and Prade, H. 2000. Decision, nonmonotonic reasoning and possibilistic logic. In Minker, J., ed., *Logic-Based Artificial Intelligence*. Kluwer Academic Publishers.
- Boutilier, C.; Brafman, R.; Hoos, H.; and Poole, D. 1999. Reasoning with conditional ceteris paribus preference statements. In *Proc. Uncertainty in Artificial Intelligence, UAI-99*.
- Boutilier, C. 1994. Towards a logic for qualitative decision theory. In *Proc. Principles of Knowledge Representation and Reasoning, KR-94*, 75–86. Morgan Kaufmann.
- Brewka, G., and Eiter, T. 1999. Preferred answer sets for extended logic programs. *Artificial Intelligence* 109:297–356.
- Brewka, G.; Benferhat, S.; and Le Berre, D. 2002. Qualitative choice logic. In *Proc. Principles of Knowledge Representation and Reasoning, KR-02*. Morgan Kaufmann.
- Brewka, G. 1996. Well-founded semantics for extended logic programs with dynamic preferences. *Journal of Artificial Intelligence Research* 4:19–36.
- Delgrande, J.; Schaub, T.; and Tompits, H. 2000. Logic programs with compiled preferences. In *Proc. European Conference on Artificial Intelligence, ECAI-00*, 464–468. IOS Press.
- Doyle, J., and Thomason, R. 1999. Background to qualitative decision theory. *AI Magazine* 20(2):55–68.
- Dubois, D.; Berre, D. L.; Prade, H.; and Sabbadin, R. 1999. Using possibilistic logic for modeling qualitative decision: Atms-based algorithms. *Fundamenta Informaticae* 34:1–30.
- Eiter, T.; Leone, N.; Mateis, C.; Pfeifer, G.; and Scarcello, F. 1998. The kr system dlv: Progress report, comparisons and benchmarks. In *Proc. Principles of Knowledge Representation and Reasoning, KR-98*. Morgan Kaufmann.
- Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing* 9:365–385.
- Grosz, B. 1999. Diplomat: Compiling prioritized rules into ordinary logic programs for e-commerce applications (intelligent systems demonstration abstract). In *Proc. AAAI-99*.
- Lang, J. 1996. Conditional desires and utilities - an alternative logical approach to qualitative decision theory. In *Proc. 12th European Conference on Artificial Intelligence, ECAI-96*, 318–322. Wiley and Sons.
- Lifschitz, V. 2001. Answer set programming and plan generation. available under [www.cs.utexas.edu/users/vl/papers.html](http://www.cs.utexas.edu/users/vl/papers.html).
- Niemelä, I., and Simons, P. 1997. Efficient implementation of the stable model and well-founded semantics for normal logic programs. In *Proc. 4th Intl. Conference on Logic Programming and Nonmonotonic Reasoning*. Springer Verlag.
- Niemelä, I., and Simons, P. 2000. Extending the smodels system with cardinality and weight constraints. In Minker, J., ed., *Logic-Based Artificial Intelligence*. Kluwer Academic Publishers.
- Poole, D. 1997. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence Journal* 94(1-2):7–56.
- Sakama, C., and Inoue, K. 1994. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning* 13:145–172.
- Schaub, T., and Wang, K. 2001. A comparative study of logic programs with preference. In *Proc. Intl. Joint Conference on Artificial Intelligence, IJCAI-01*.
- Soininen, T. 2000. *An Approach to Knowledge Representation and Reasoning for Product Configuration Tasks*. Ph.D. Dissertation, Helsinki University of Technology, Finland.
- van der Torre, L., and Weydert, E. 2001. Parameters for utilitarian desires in a qualitative decision theory. *Applied Intelligence* 14:285–301.