Implementing Ordered Choice Logic Programming using Answer Set Solvers

Marina De Vos[*]

Department of Computer Science
University of Bath
Bath, United Kingdom
mdv@cs.bath.ac.uk

Abstract. Ordered Choice Logic Programming (OCLP) allows for dynamic preference-based decision-making with multiple alternatives without the need for any form of negation. This complete absence of negation does not weaken the language as both forms (classical and as-failure) can be intuitively simulated in the language and eliminated using a simple pre-processor, making it also an easy language for users less familiar with logic programming. The semantics of the language is based on the preference between alternatives, yielding both a skeptical and a credulous approach. In this paper we demonstrate how OCLPs can be translated to semi-negative logic programs such that, depending on the transformation, the answer sets of the latter correspond with the skeptical/credulous answer sets of the former. By providing such a mapping, we have a mechanism for implementing OCLP using answer set solvers like Smodels or dlv. We end with a discussion of the complexity of our system and the reasoning tasks it can perform.

1 Introduction

Examining human reasoning, we find that people often use preference, order or defaults for making decisions: “I prefer this dish”, “This colour goes better with the interior”, “This item costs more”, “In general, the human heart is positioned at the left”. When faced with conflicting information, one tends to make decisions that prefer an alternative corresponding to more reliable, more complete, more preferred or more specific information. When modelling knowledge or non-monotonic reasoning via computer programs, it is only natural to incorporate such mechanisms.

In recent years several proposals for the explicit representation of preference in logic programming formalisms have been put forward. [19, 18] are just two examples.

Systems that support preferences find applications in various domains such as law, object orientation, scheduling, model based diagnosis and configuration tasks. However, most approaches use preferences only when the models have already been computed, i.e. decisions have already been made, or only support preferences between rules with

[*] This work was partially funded by the Information Society Technologies programme of the European Commission, Future and Emerging technologies under the IST-2001-37004 WASP project.

opposite (contradictory) consequences, thus statically limiting the number of alternatives of a decision.

We propose a formalism ([16]), called ordered choice logic programming, that enables one to dynamically reason about situation-dependent decisions involving multiple alternatives. The dynamics of this system is demonstrated by the following example.

Example 1. Buying a laptop computer involves a compromise between what is desirable and what is affordable. Take, for example, the choice between a CD, CDRW or DVD drive. The CD is the cheaper option. On the other hand, for a laptop, a DVD drive may be more useful than a CD writer. If the budget is large enough, one could even buy two of the devices. The above information leads one to consider two possible situations.

– With a smaller budget, a DVD-player is indicated, while
– with a larger budget, one can order both a DVD-player and a CD-writer.

To allow this kind of reasoning, a program consists of a (strict) partially ordered set of components containing choice rules (rules with exclusive disjunction in the head). Information flows from less specific components to the more preferred ones until a conflict among alternatives arises, in which case the most specific one will be favoured. The situation becomes less clear when two alternatives are equally valued or are unrelated. The decision in this case is very situation dependent: a doctor having a choice between two equally effective cures has to make a decision, while you better remain indecisive when two of your friends have an argument! To allow both types of intuitive reasoning, two semantics are introduced: a credulous and a more skeptical one.

OCLP provides an elegant and intuitive way of representing and dealing with decisions. People with little or no experience with non-monotonic reasoning can easily relate to it, due to the absence of negation. This absence of negation does not restrict the language in any way, as both types (classic and as-failure) can easily be simulated. When users insist they can use negation, a simple pre-processor can then be used to remove it while maintaining the semantics.

In computer science, having a nice theory alone is not enough; one also needs to be able to apply it. The aim of this paper is to provide the theoretical foundations for an implementation.

In this paper, we investigate the possibility for building an OCLP front-end for answer set solvers. Smodels ([20]), developed at Helsinki University of Technology, and dlv ([26]), created at the Technical University of Vienna and the University of Calabria are currently the most popular ones. An initial implementation build on top of Smodels can be obtained from http://www.cs.bath.ac.uk/∼mdv/oct/.

The remainder of this paper is organised as follows: in Section2 we continue with a short overview of the basic notions concerning choice logic programming, the language behind OCLP. Section 3 focuses on the introduction of OCLP with its skeptical and credulous answer set semantics. Section 4 deals with two mapping of OCLPs to semi-negative logic programs allowing answer set solvers to work with OCLP. We also take a closer look at the complexity of the proposed mappings. These transformations, one for each semantics, can then serve as a theoretical model for our front-end. We investigate various ways how we could, implementation wise, improve them. Furthermore, we introduce an initial implementation, called OCT, which computes both types of answer sets on top of Smodels. When explaining the theoretical aspects of both OCLP and the mappings we always considered our programs to be grounded. At the end of the section, we have a closer look at the technical issues that arise when we extend to the non-grounded case with a finite Herbrand Universe. In Section 6, we investigate the complexity and the expressive power of our language. We end this paper with a discussion on the relations with other approaches (Section 7) and directions for future research (Section 8).

2 Choice Logic Programming

Choice logic programs [15] represent decisions by interpreting the head of a rule as an exclusive choice between alternatives.

Formally, a *Choice Logic Program* [15], CLP for short, is a countable set of rules of the form $A \leftarrow B$ where A and B are finite sets of ground atoms. Intuitively, atoms in A are assumed to be xor'ed together while B is read as a conjunction (note that A may be empty, i.e. constraints are allowed). The set A is called the head of the rule r, denoted H_r, while B is its body, denoted B_r. In examples, we use "$\oplus$" to denote exclusive disjunction[1], while "," is used to denote conjunction.

The *Herbrand base* of a CLP P, denoted $\mathcal{B}_P$, is the set of all atoms that appear in P. An *interpretation*[2] is a subset of $\mathcal{B}_P$.

A rule r in a CLP is said to be *applicable* w.r.t. an interpretation I if $B_r \subseteq I$. Since we are modelling choice, we have that r is *applied* when r is applicable and[3] $|H_r \cap I| = 1$. A rule is *satisfied* if it is applied or not applicable. A *model* is defined in the usual way as a total interpretation that satisfies every rule. A model M is said to be *minimal* if there does not exist a model N such that $N \subset M$.

3 Ordered Choice Logic Programming

An ordered choice logic program (OCLP) is a collection of choice logic programs, called components, which are organised in a strict partial order[4] that represents some preference criterion (e.g. specificity, reliability, ...).

Definition 1. *An* ***Ordered Choice Logic Program****, or OCLP, is a pair $\langle \mathcal{C}, \prec \rangle$ where $\mathcal{C}$ is a finite set of choice logic programs, called* ***components****, and "$\prec$" is a strict pointed partial order on $\mathcal{C}$.*

[1] The exclusive disjunction used here is stronger than the one used in classic propositional logic. Here we assume that $a \oplus b \oplus c$ can only be true iff exactly one atom is true. The same statement in classical logic is also true when all three atoms are true.

[2] In this paper we only work with total interpretations: each atom from the Herbrand base is either true or false. Bearing this in mind, it suffices to mention only those atoms which can be considered true.

[3] For a set X, we use $|X|$ do denote its cardinality.

[4] A relation R on a set A is a strict partial order iff R is anti-reflexive, anti-symmetric and transitive. R is pointed if an element $a \in A$ exists such that aRb for all $b \in A$.

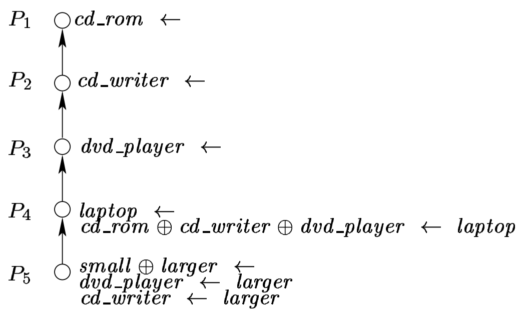
P_1 $cd\_rom \leftarrow$
P_2 $cd\_writer \leftarrow$
P_3 $dvd\_player \leftarrow$
P_4 $laptop \leftarrow$
$cd\_rom \oplus cd\_writer \oplus dvd\_player \leftarrow laptop$
P_5 $small \oplus larger \leftarrow$
$dvd\_player \leftarrow larger$
$cd\_writer \leftarrow larger$

Fig. 1. The Configuration OCLP of Example 2

For two components $C_1, C_2 \in \mathcal{C}$, $C_1 \prec C_2$ implies that C_1 is preferred over C_2. Throughout the examples, we will often represent an OCLP P by means of a directed acyclic graph (dag) in which the nodes represent the components and the arcs the $\prec$-relation, where arcs point from smaller (more preferred) to larger (less preferred) components.

Example 2. The decision problem from the introduction (Example 1) can easily be written as an OCLP, as shown in Figure 1. The rules in components P_1, P_2 and P_3 express the preferences in case of a small budget. The rules in P_4 express the intention to buy/configure a laptop and, because of this, a decision about its various devices should be made. In component P_5, the first rule states the possibility of a larger budget. If so, the two remaining rules allow the purchase of both a DVD-player and a CD-writer.

Definition 2. *Let P be an OCLP. We use $P^\star$ to denote the CLP that contains all the rules appearing in (a component of) P. We assume that rules in $P^\star$ are labelled by the component from which they originate and we use $c(r)$ to denote the component*[5] *of r. The* Herbrand base *$\mathcal{B}_P$ of P is defined by $\mathcal{B}_P = \mathcal{B}_{P^\star}$.*
An ***interpretation*** *for P is any interpretation of $P^\star$. We say that a rule r in P is* ***applicable*** *w.r.t. an interpretation I iff $B_r \subseteq I$; r is* ***applied*** *w.r.t. I iff r is applicable and $|H_r \cap I| = 1$.*

Example 3. For the OCLP in Example 2, the sets $I = \{dvd\_player, small\}$, $J = \{laptop, cd\_writer, small\}$, $K = \{laptop, dvd\_player, small\}$ and $L = \{dvd\_player, larger, cd\_writer, cd\_player, laptop\}$ are all interpretations. The interpretation I makes the rule $small \oplus larger \leftarrow$ applied while the applicable rule $cd\_writer \leftarrow$ is not applied.

Facing a decision means making an exclusive choice between the various alternatives which are available. If we want OCLP to model/solve decision problems we need a mechanism for representing them. In a CLP, decisions are generated by so-called *choice rules* i.e. rules with multiple head atoms. For OCLP, we can do something similar as long as we also take the preference order into account. We want to make sure that we

[5] Without losing generality, we can assume that a rule appears in only one component.

leave the option open to overrule the exclusiveness of a choice when in more preferred components multiple alternatives are suggested (e.g. Example 1). Hence we say that an atom a is an *alternative* for an atom b in a component C if an applicable rule exists in a component at least as preferred as C containing both a and b in its head.

Definition 3. *Let I be an interpretation of an OCLP $P = \langle \mathcal{C}, \prec \rangle$ with $C \in \mathcal{C}$. The set of* ***alternatives*** *in C for an atom $a \in \mathcal{B}_P$ w.r.t. I, denoted $\Omega^I_C(a)$, is defined as*[6]*: $\Omega^I_C(a) = \{b \mid \exists r \in P^\star \cdot c(r) \preccurlyeq C \ \wedge\ B_r \subseteq I \ \wedge\ a, b \in H_r$ with $a \neq b\}$.*

Example 4. Reconsider Example 3. The alternatives for cd_rom in P_2 w.r.t. J are $\Omega^J_{P_2}(cd\_rom) = \{dvd\_player, cd\_writer\}$. W.r.t. I, we obtain $\Omega^I_{P_2}(cd\_rom) = \emptyset$, since the choice rule in P_4 is not applicable. When we take P_5 instead of P_2, we obtain w.r.t. J: $\Omega^J_{P_5}(cd\_rom) = \emptyset$.

Given the alternatives in a certain context (a component and an interpretation), one naturally selects that alternative that is motivated by a more preferred rule, thus *defeating* the rule(s) suggesting less preferred alternatives. However, if alternatives appear in the same or unrelated components, two approaches are possible: using a skeptical strategy, one would refrain from making a decision, i.e. not selecting any of the various alternatives, while a credulous setting suggests an arbitrary choice of one of the alternatives. For both types of reasoning one can think of situations where one approach works while the other gives an incorrect, unintuitive outcome. Skeptical reasoning is practiced in American law when a jury cannot come to a unanimous decision. An example of credulous reasoning is the decision a goal-keeper faces in football when trying to stop a penalty. To accommodate this problem, we introduce a semantics for both types of reasoning. From a skeptical viewpoint, we say that rule is defeated if one can find a better, more preferred alternative for each of its head atoms.

Definition 4. *Let I be an interpretation for an OCLP P. A rule $r \in P^\star$ is* ***defeated*** *w.r.t. I iff $\forall a \in H_r \cdot \exists r' \in P^\star \cdot c(r') \prec c(r) \ \wedge\ B_{r'} \subseteq I \ \wedge\ H_{r'} \subseteq \Omega^I_{c(r)}(a)$.*

Example 5. Reconsider Example 3. The rule $cd\_rom \leftarrow$ is defeated w.r.t. J by the rule $cd\_writer \leftarrow$. The rule $cd\_rom \oplus cd\_writer \oplus dvd\_player \leftarrow$ is defeated w.r.t. L by the combination of the rules $dvd\_player \leftarrow larger$ and $cd\_writer \leftarrow larger$.

Example 6. Consider the OCLP $\langle \{P_1 = \{a \leftarrow ; b \leftarrow\}, P_2 = \{a \oplus b \leftarrow\}\}, P_2 \prec P_1 \rangle$. Given the interpretation $\{b\}$, the rule $r : a \leftarrow$ is not defeated. The atom a has one alternative, i.e. b, thanks to the applicable choice rule in component P_2. However, examining the more preferred components of $c(r) = P_1$, we cannot find any rule having b in the head.

Just as for the skeptical semantics we need to define an appropriate defeating strategy for our credulous approach. An obvious way of doing so consists of simply dropping the condition that an alternative should be found in a more preferred component. Unfortunately, this leads to unintuitive results. To avoid this, we need to make sure that credulous defeaters are not only applicable, but also applied.

[6] $\preccurlyeq$ is the reflexive closure of $\prec$.

Definition 5. *Let I be an interpretation for an OCLP P. A rule $r \in P^\star$ is **c-defeated** w.r.t. I iff $\forall a \in H_r \cdot \exists r' \in P^\star \cdot c(r) \not\prec c(r') \wedge r'$ is applied w.r.t. $I \wedge H_{r'} \subseteq \Omega^I_{c(r)}(a)$.*

Note that this definition allows the rules r' to come from a more preferred, the same or unrelated component. At first sight one might think that a situation could occur where $r = r'$. However, this is impossible as an atom can never be considered an alternative of itself.

Example 7. While the skeptical approach makes it impossible to have the rule $a \leftarrow$ in Example 6 defeated w.r.t. $\{b\}$, the credulous semantics can.

For our model semantics, both skeptical as credulous, rules that are not satisfied (as for choice logic programs) must be (c-)defeated.

Definition 6. *Let P be an OCLP. A total interpretation I is a **skeptical/credulous model** iff every rule in $P^\star$ is either not applicable, applied or (c-)defeated w.r.t. I. A skeptical/credulous model M is **minimal** iff M is minimal according to set inclusion, i.e. no skeptical/credulous model N of P exists such that $N \subset M$.*

Example 8. Reconsider the interpretations I, J, K and L from Example 3. Only K and L are skeptical/credulous models. Model L is not minimal due to the skeptical/credulous model $Z = \{dvd\_player, cd\_writer, laptop, larger\}$. The minimal skeptical/credulous models K and Z correspond to the intuitive outcomes of the problem.

Example 9. The program of Example 6 has no skeptical models but two credulous ones: $\{a\}$, $\{b\}$.

The next example illustrates that the skeptical/credulous model semantics does not always provide the appropriate solutions to the decision problem at hand.

Example 10. Consider the ordered choice logic program $P = \langle\{P_1 = \{a \leftarrow\}, P_2 = \{b \leftarrow\}, P_3 = \{a \oplus b \leftarrow c\}, P_3 \prec P_2 \prec P_1\rangle$, where P has two minimal skeptical/credulous models: $M = \{b, c\}$, and $N = \{a, b\}$. Clearly, c is an unsupported assumption in M, causing P_3 to trigger an unwarranted choice between a and b.

We introduce an adaptation of the Gelfond-Lifschitz [23] and reduct ([25]) transformations to filter unintended (minimal) models containing unsupported atoms. This results in the skeptical/credulous answer set semantics.

Definition 7. *Let M be a total interpretation for an OCLP P. The **Gelfond-Lifschitz transformation** (resp. **reduct**) for P w.r.t. M, denoted P^M (resp. P^M_c), is the CLP obtained from $P^\star$ by removing all (c-)defeated rules. M is called a **skeptical (resp. credulous) answer set** for P iff M is a minimal model[7] for P^M (resp. P^M_c).*

Although both answer set semantics produce models (skeptical or credulous ones) for the program, they differ in whether they produce minimal ones or not. Just as for answer sets of semi-negative logic programs, we find that skeptical answer sets are minimal skeptical models. For extended disjunctive logic programs, the answer set semantics is not minimal[25]. The same applies for credulous answer sets of ordered choice logic programs, as demonstrated by the following example.

[7] The definition in [16] states a stable model, but since both are identical for CLP, we have opted in this paper to use minimal model instead.

Example 11. Consider the program $P = \langle\{P_1 = \{r_1 : g \leftarrow\}, P_2 = \{r_2 : p \oplus d \leftarrow; r_3 : g \oplus p \leftarrow; r_4 : g \oplus d \leftarrow\}\}, P_2 \prec P_1\rangle$. Consider $M_1 = \{g\}$ and $M_2 = \{g, d\}$. Clearly, $M_1^+ \subset M_2^+$, while both interpretations are credulous answer sets for P. For M_1, we have that $P_c^{M_1} = \{g \leftarrow; g \oplus d \leftarrow; g \oplus p \leftarrow\}$ for which it can easily be verified that M_1 is a minimal model. The program $P_c^{M_2} = \{p \oplus d \leftarrow ; g \oplus p \leftarrow\}$ has two minimal models: $\{p\}$ and $\{g, d\}$. Note that M_2 is a credulous model because the c-defeater has become c-defeated, i.e. the justification in M_1 for c-defeating $p \oplus d \leftarrow$ has disappeared in M_2.

Non-minimal credulous answer sets appear when the program contains inconsistencies on a decision level: in the above example the following choices have to be made: $\{p, d\}$, $\{g, p\}$ and $\{g, d\}$. Because of the program's construction, one can choose either one or two alternatives and c-defeating will make the choice justifiable.

4 Implementation

For the last five years, answer set programming has gained popularity. One of the main forces behind this is the growing efficiency of answer solvers like smodels ([20]) and dlv ([26]).

In this section, we propose a mapping, for both semantics, to semi-negative logic programs. Since both answer set solvers support this type of programs, this would allow us to implement an OCLP front-end for them. In this section we will present an initial implementation designed to work on top of smodels. So far our efforts have been restricted to the grounded case, in the last part of this section we extend our approach to non-grounded programs.

The skeptical answer set semantics is based on the notion of defeat. If we want to map our formalism to a language which does not support this, we need a way to encode it. This implies anticipating which combinations of rules could be capable of defeating a rule and which ones are not.

The definition of defeating relies strongly on the notion of alternatives: rules can only be defeated by rules containing alternatives of the head atoms. Therefore, anticipating defeaters also implies predicting alternatives. According to Definition 3, b is an alternative of a in a component C if one can find an applicable choice rule as preferred as C containing both a and b in the head. This implies that even without an interpretation we can find out which atoms might be or could become alternatives; it only remains to be checked if the rule is applicable or not. These condition-based alternatives are referred to as possible future alternatives and are defined more formally below.

Definition 8. *Let P be an OCLP, $C \in \mathcal{C}$ be component of P and $a \in \mathcal{B}_P$. The set of* ***possible future alternatives*** *of a in C, denoted as $\mathcal{A}_C^P(a)$, is defined as $\mathcal{A}_C^P(a) = \{(b, B_r) \mid \exists r \in P \cdot c(r) \preccurlyeq C, a, b \in H_r, a \neq b\}$.*

Example 12. Consider the OCLP $P = \langle\{P_1 = \{r_1 : a \leftarrow; r_2 : f \leftarrow\}, P_2 = \{r_3 : a \oplus b \oplus c \leftarrow d; r_4 : a \oplus d \leftarrow f; r_5 : d \oplus c \leftarrow\}, P_2 \prec P_1\}\rangle$. The possible future alternatives of a in P_1 equal $\mathcal{A}_{P_1}^P(a) = \{(b, \{d\}), (c, \{d\}), (d, \{f\})\}$.

It is easy to see that one can compute the possible future defeaters in one iteration of the program, making the process polynomial.

The next theorem demonstrates that alternatives can be expressed in terms of possible future alternatives.

Theorem 1. *Let P be an OCLP, $C \in \mathcal{C}$ be component of P, $a \in \mathcal{B}_P$ and I an interpretation for P. Then, $\Omega^I_C(a) = \{b \mid (b, S) \in \mathcal{A}^P_C(a) \wedge S \subseteq I\}$.*

Having these possible future alternatives allows us to detect possible future defeaters in much the same way as we detect standard defeaters (Definition 4). The only extra bit we need is to collect all the conditions on the alternatives. This collection then acts as the condition for the defeating rule.

Definition 9. *Let P be an OCLP, $C \in \mathcal{C}$ be component of P and $a \in \mathcal{B}_P$. The set of **possible future defeaters** of a in C, denoted as $\mathcal{D}^P_C(a)$, is defined as $\mathcal{D}^P_C(a) = \{(r, S) \mid \exists r \in P \cdot c(r) \prec C, \forall b \in H_r \cdot (b, B_b) \in \mathcal{A}^P_C(a), S = \bigcup_{b \in H_r} B_b\}$. The set of **possible future defeaters** of a rule $r \in P$, denoted as $\mathcal{D}^P(r)$, is defined as $\mathcal{D}^P(r) = \{(R, S) \mid S = \bigcup_{a \in H_r} S_a$ such that $(r_a, S_a) \in \mathcal{D}^P_{c(r)}(a), r_a \in R\}$.*

Having the possible future defeaters of an atom in a certain component, we can easily find that combination that can act as a possible future defeater of a rule in a certain component. We simply compute the set of possible future defeaters of each of the head atoms of this rule in this rule's component. The set of all possible permutations of choosing an element from each of these sets give us the possible future defeaters of our rule. In other words, we obtain a number of possible future defeaters of a rule equal to the product of the sizes of the sets of possible future defeaters for each of its head elements.

Example 13. When we look back to the program P of Example 12, we have that a has a one possible future defeater in P_1 as: $\mathcal{D}^P_{P_1}(a) = \{(r_5, \{d, f\})\}$. In the same component, we have that c has a future defeater $\mathcal{D}^P_{P_1}(c) = \{(r_4, \{d, f\})\}$. All the other atoms in the program do not have any possible future defeaters in any of the relevant components. The rule r_1 is the only rule with possible future defeaters, namely $\mathcal{D}^P(r_1) = \{(\{r_5\}, \{d, f\})\}$.

Computing all possible defeaters of an atom can be done by one pass of the program. Therefore, computing possibly defeaters of all atoms can be done in polynomial time. The possible future defeaters of a rule can be obtained in polynomial time. By their construction, the number of possible future defeaters is polynomial with respect to number of atoms in the program.

Clearly, possible future defeaters can be used for expressing interpretation-dependent defeaters.

Proposition 1. *Let P be an OCLP and let I be an interpretation for it. A rule $r \in P^\star$ is defeated w.r.t. I iff $\exists (R, S) \in \mathcal{D}^P(r) \cdot S \subseteq I, B_{r'} \subseteq I, \forall r' \in R$.*

These possible future defeaters are the key to mapping OCLPs to semi-negative logic programs. We are only required to turn the information which makes possible future defeaters into defeaters, i.e. they have to be applicable, into a condition. To make this possible, we introduce for each non-constraint rule r in the program two new atoms: d_r and a_r. The former indicates that the rule r is defeated or not, while the truth value of the latter is an indicator of the applicability of the rule.

Definition 10. *Let P be an OCLP. Then, the logic program*[8] $P_\neg$ *is defined as follows:*

1. $|H_r| = 0$*:* $r \in P_\neg$
2. $|H_r| \geq 1$*:*
   (a) $h \leftarrow B_r, \neg d_r, \neg(H_r \setminus \{h\}) \ \in P_\neg$*:* $\forall h \in H_r$
   (b) $a_r \leftarrow B_r \ \in P_\neg$
   (c) $d_r \leftarrow C \ \in P_\neg$ *with* $C = S \cup \bigcup_{r' \in R} a_{r'}$ *such that* $(R, S) \in \mathcal{D}^P(r)$.
   (d) $\leftarrow h, g, B_r, \neg d_r \ \in P_\neg$*:* $\forall h, g \in H_r \cdot h \neq g$

Since constraints are not involved in the defeating process, we can simply copy them to the corresponding logic program. For the answer set semantics of ordered choice logic program, we need, among other things, that each applicable, undefeated rule admits exactly one head atom. Rules of type a) and d) make sure that the corresponding rules in the logic program do not violate this property. The rules of type b) indicate which original rules are applicable. The c)-rules are probably the most difficult ones. They express when a rule should or could be considered defeated. If we look at Theorem 1, we have a mechanism for relating possible future defeaters to actual defeaters. Given a possible future defeater (R, S) for a rule r , we simply have to make sure that all rules in R are applicable and that all atoms in S are true with respect to the current interpretation. With rules of type b), we can express the former using a_r. Combining all of this, we can signal in the transformed program that a rule is defeated or not using a rule $d_r \leftarrow a_{r_1}, \ldots, a_{r_n}, S$ with $r_i \in R$ and $n = |H_r|$. Whenever an answer set of the transformed program makes d_r true, we know that the original rule r is defeated. The construction with rules of type b) makes sure that the reverse also holds.

Example 14. The corresponding logic program $P_\neg$ of the OCLP of Example 12 looks like:

$$\begin{array}{llll}
a \leftarrow \neg d_{r_1} & a \leftarrow f, \neg d, \neg d_{r_4} & a_{r_2} \leftarrow & \leftarrow d, \neg d_{r_3}, a, b \\
f \leftarrow \neg d_{r_2} & d \leftarrow f, \neg a, \neg d_{r_4} & a_{r_3} \leftarrow d & \leftarrow d, \neg d_{r_3}, a, c \\
a \leftarrow d, \neg b, \neg c, \neg d_{r_3} & d \leftarrow \neg c, \neg d_{r_5} & a_{r_4} \leftarrow f & \leftarrow d, \neg d_{r_3}, b, c \\
b \leftarrow d, \neg a, \neg c, \neg d_{r_3} & c \leftarrow \neg d, \neg d_{r_5} & a_{r_5} \leftarrow & \leftarrow f, \neg d_{r_4}, a, d \\
c \leftarrow d, \neg a, \neg b, \neg d_{r_3} & a_{r_1} \leftarrow & d_{r_1} \leftarrow a_{r_5}, d, f & \leftarrow \neg d_{r_5}, d, c
\end{array}$$

The original OCLP of Example 12 has two skeptical answer sets, $\{f, d, b\}$ and $\{f, c, a\}$, which correspond exactly with the two answer sets, $\{a_{r_1}, a_{r_2}, a_{r_3}, a_{r_4}, a_{r_5}, d_{r_1}, f, d, b\}$ and $\{a_{r_1}, a_{r_2}, a_{r_4}, a_{r_5}, f, c, a\}$, of $P_\neg$.

Because the transformation is based on possible future defeaters, it is easy to see that the transformation can be constructed in polynomial time, leading to only a polynomial increase of rules.

[8] In this paper, we use $\neg$ to represent negation as failure.

Theorem 2. *Let P be an OCLP and $P_\neg$ be its corresponding logic program. Then, a one-to-one mapping exists between the skeptical answer sets M of P and the answer sets N of $P_\neg$ in such a way that $N = M \cup \{a_r \mid \exists r \in P \cdot |H_r| \geq 1, B_r \subseteq M\} \cup \{d_r \mid \exists r \in P \cdot r$ is defeated w.r.t. $M\}$.*

4.1 Credulous Mapping

To obtain the credulous answer set semantics for OCLPs, we propose a similar mapping to semi-negative logic programs. The only difference between the skeptical and the credulous semantics is the way they both handle defeat. For the credulous version, we need to make sure that we look for c-defeaters in all components which are not less preferred as the rule we wish to defeat. Furthermore, we have to make sure that c-defeaters are applied and not just applicable as is the case for defeaters. The former will be encoded by means of possible future c-defeaters while the latter will be translated in a different style of a_r rules in the mapping.

The definition of possible future c-defeater is identical to the one of its skeptical counter-part except that it looks for rules in all components which are not less preferred.

Definition 11. *Let P be an OCLP, $C \in \mathcal{C}$ be component of P and $a \in \mathcal{B}_P$. The set of* ***possible future c-defeaters*** *of a in C, denoted as $\mathcal{F}_C^P(a)$, is defined as $\mathcal{F}_C^P(a) = \{(r, S) \mid \exists r \in P \cdot C \not\prec c(r), \forall b \in H_r \cdot (b, B_b) \in \mathcal{A}_C^P(a), S = \bigcup B_b\}$. The set of* ***possible future c-defeaters*** *of a rule $r \in P$, denoted as $\mathcal{F}^P(r)$, is defined as $\mathcal{F}^P(r) = \{(R, S) \mid S = \bigcup_{a \in H_r} S_a$ such that $(r_a, S_a) \in \mathcal{F}_{c(r)}^P(a), r_a \in R\}$.*

Just as their sceptical counterparts, possible future c-defeaters can be obtained in polynomial time. Their number is also polynomial w.r.t. the number of atoms in the program.

Just as before, c-defeaters can be expressed in terms of possible future c-defeaters.

Theorem 3. *Let P be an OCLP and let I be an interpretation for it. A rule $r \in P^\star$ is c-defeated w.r.t. I iff $\exists (R, S) \in \mathcal{F}_{c(r)}^P(a) \cdot S \subseteq I, r'$ applied w.r.t. $I, \forall r' \in R$.*

Definition 12. *Let P be an OCLP. Then, the logic program $P_\neg^c$ is defined as follows:*
1. *$|H_r| = 0$: $r \in P_\neg^c$*
2. *$|H_r| \geq 1$:*
   (a) *$h \leftarrow B_r, \neg d_r, \neg(H_r \setminus \{h\}) \ \in P_\neg^c$: $\forall h \in H_r$*
   (b) *$a_r \leftarrow B_r, h, \neg(H_r \setminus \{h\}) \ \in P_\neg^c$: $\forall a \in H_r$*
   (c) *$d_r \leftarrow C \ \in P_\neg^c$ with $C = S \cup \bigcup_{r' \in R} a_{r'}$ with $(R, S) \in \mathcal{F}^P(r)$.*

The credulous mapping is very similar to the skeptical one but there are a couple of subtle differences: an obvious difference is the use of possible future c-defeater instead of their skeptical counterparts (c-rules). The second change are the rules implying a_r (b-rules). Previously they were used to indicate applicability, the necessary condition for the defeat. Since c-defeat works with applied defeaters, we need to make sure that a_r is considered only true when r is applied. The less obvious change is the absence of the rules of type d). Since a rule can only be applied when one and only one head atom is considered true and because a_r should only be considered true in this particular case, they no longer necessary.

This transformation can also be performed in polynomial time and results in a polynomial increase of rules.

Example 15. Reconsider the OCLP from Example 11. If we use the mapping from Definition 12, we obtain the following program:

$$\begin{array}{lll}
g \leftarrow \neg d_1 & a_1 \leftarrow g & d_1 \leftarrow a_2 \\
p \leftarrow \neg d, \neg d_2 & a_2 \leftarrow p, \neg d & d_2 \leftarrow a_3, a_4 \\
d \leftarrow \neg p, \neg d_2 & a_2 \leftarrow d, \neg p & d_3 \leftarrow a_2, a_4 \\
g \leftarrow \neg p, \neg d_3 & a_3 \leftarrow g, \neg p & d_4 \leftarrow a_2, a_3 \\
p \leftarrow \neg g, \neg d_3 & a_3 \leftarrow p, \neg g & \\
g \leftarrow \neg d, \neg d_4 & a_4 \leftarrow g, \neg d & \\
d \leftarrow \neg g, \neg d_4 & a_4 \leftarrow d, \neg g &
\end{array}$$

The answer sets of this program correspond perfectly to the credulous answer sets of the original program. The newly introduced atoms make sure that the answer set semantics remains minimal while the credulous OCLP version is clearly not.

Theorem 4. *Let P be an OCLP and $P_\neg$ be its corresponding logic program. Then, a one-to-one mapping exists between the credulous answer sets M of P and the answer sets N of $P_\neg$ in such a way that $N = M \cup \{a_r \mid \exists r \in P \cdot |H_r| \geq 1, B_r \subseteq M, |H_r \cap M| = 1\} \cup \{d_r \mid \exists r \in P \cdot r$ is c-defeated w.r.t. $M\}$.*

4.2 Using Answer Set Solvers

The transformations presented above give us the theoretical certainty that OCLP can be implemented on top of answer set solvers like Smodels ([20]), and dlv ([26]).

However, no attempt is made to make the mappings efficient, apart from assuring that they can be done in polynomial time. When actually implementing the system, we should take efficiency, both time as space wise, into account. New atoms and rules should only be created when necessary.

For example, if one of the head atoms of a rule does not have any alternatives relative to the component of the rule, there is hardly any point of mentioning d_r, as no rule will be generated anyway. The same applies for the a_r rules: if a rule does not stand the chance to be a (c-)defeater, then there is no point in defining it. Theoretically speaking, there is no need to introduce the atoms a_r. One can easily incorporate the bodies into the rules describing the defeating conditions. Unfortunately, this makes the mapping harder to read and would, in the credulous case, create more rules of type d). For each defeater, one has to create rules to accommodate all the various ways this rule can be made applied. By using a_r, one only needs to do this once, while without it one has to do this over and over again. For larger and more complex programs this can cause a serious overhead. In algorithmic form of the mappings, it would be best to start with generating the c)-rules. By doing so, one can immediate tell which a_r rules are necessary and whether $notd_r$ should be included in the corresponding rule or not.

An other time/space saving point is the effective usage of the various constructs provided by the answer set solver at hand. The disjunctive rules used by dlv can help us to reduce the number of rules, by simply replacing the a)-rules with a single disjunctive rule. Unfortunately, this measure does not eliminate the need for the constraints, assuring that each applicable rule is made applied by a single true head atom. Smodels on the other hand provides us with a mechanism of writing all shifted rules and the corresponding constraints as a single rule using their special choice construct.

Example 16. Reconsider the OCLP mentioned in Example 12. If we only take into account the special construct provided my smodels, we obtain the following program:

```
a :- not dr1.                     ar2.
f :- not dr2.                     ar3 :- 1{a,b,c}1.
1{a,b,c}1 :- d, not dr3.          ar4 :- 1{a,d}1.
1{a,d}1 :- f, not dr4.            ar5 :- 1{d,c}1.
1{d,c}1 :- not dr5.               dr1 :- ar5, d, f.
ar1.
```

By taken all the other rule and atom saving measures into account, we obtain:

```
a :- not dr1.                     1{d,c}1
f.                                ar5 :- 1{d,c}1.
1{a,b,c}1 :- d.                   dr1 :- ar5, d, f.
1{a,d}1 :- f
```

4.3 Implementation an OCLP Front-end to Smodels

To demonstrate the theoretical mapping and to serve as a basis for future experiments and research a simple language was developed to allow OCLP to be processed by computer. A compiler[9] was created to parse the input language and interface into the Smodels([20]) API which was then used to compute the answer sets. The compiler *OCT* is available under the GPL (“open source”) from http://www.cs.bath.ac.uk/∼mdv/oct/.

Example 17. The OCLP from Figure 2 can be very easily written as an input file for the oct-compiler:

```
component c01 {  a.  }

component c02 {  a + b :- d. }

component c03 {  b.  }

component c04 {  d + c. }

c04 < c03 < c02 < c01
```

[9] Here compiler is used in the broader sense of an automated computer language translation system rather than traditional procedural to machine code system.

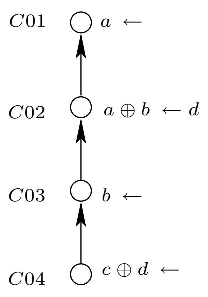

Fig. 2. The OCT program of Example 17

Definitions 10 and 12 give us a theoretical basis for a program to convert OCLPs in semi-negative logic programs but a few changes and optimisations are necessary before we have an effective algorithm for converting and solving OCLPs. All optimisations available in oct are based on the information obtained from the mapping and none use any special constructs provided by Smodels. This allows for the usage of OCT on top of any other answer set solver. Two types of optimisations are provided: intra-transform and inter-transform. The former are carried out during the transformation of OCLP to LP on a one rule basis, the latter are recursively applied between rules after the initial mapping. More information on the various information techniques can be found in [5].

Example 18. If we reconsider the OCLP from Example 17, OCT without optimisations provides us with the following semi-negative logic program:

```
  ⎡ oclp_a_0 .
0 ⎢ a :- not oclp_d_0 .
  ⎣ oclp_d_0 :- d , oclp_a_2 .
  ⎡ oclp_a_1 :- d .
1 ⎢ a :- not oclp_d_1 , not b , d .
  ⎢ :- not oclp_d_1 , a , b , d .
  ⎣ b :- not oclp_d_1 , not a , d .
2 ⎡ oclp_a_2 .
  ⎣ b :- not oclp_d_2 .
  ⎡ oclp_a_3 .
3 ⎢ d :- not oclp_d_3 , not c .
  ⎢ :- not oclp_d_3 , d , c .
  ⎣ c :- not oclp_d_3 , not d .
```

When we allow for optimisations, we obtain a much smaller and compacter program:

```
a :- not d .
:- a , d .
b .
d :- not c .
:- d , c .
c :- not d .
```

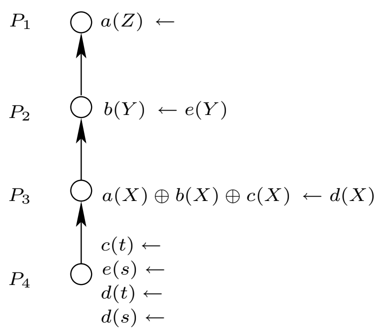

Fig. 3. The non-grounded OCLP of Example 19

5 The Non-Grounded Case

So far we have always assumed that all our programs were fully grounded, i.e. all rules have been replaced by their ground instances obtained from substituting all variables with constants appearing in the program. This process of grounding has no influence on the semantics of our programs. Using variables makes the program more compact and readable. All our definitions and theorems remain valid in the presence of variables and a countable Herbrand Universe. However, writing them down in a reader-friendly manner becomes more complicated since at various stages variables have to be renamed to avoid confusion and/or incorrect grounding in future stages. Furthermore, the atoms a_r and d_r need to be equipped with variables to make sure that only particular groundings are considered for applicability and (c-)defeat.

Let us demonstrate our approach with an example.

Example 19. Consider the non-grounded OCLP in Figure 3. Clearly in this example, $a(X)$, $b(Y)$ and $c(Z)$ can only be alternatives if $d(X)$ is true for some X and if they all have the same substitution. This will be taken into account when obtaining possible future alternatives and possible future (c-)defeaters. This is also reflected in the variables we add to a_r and d_r. For our program, we obtain:

$a(Z) \leftarrow \neg d_{r_1}(Z)$
$b(Y) \leftarrow \neg d_{r_2}(Z)$
$a(X) \leftarrow d(X), \neg d_{r_3}(X), \neg b(X), \neg c(X)$
$b(X) \leftarrow d(X), \neg d_{r_3}(X), \neg a(X), \neg c(X)$
$c(X) \leftarrow d(X), \neg d_{r_3}(X), \neg a(X), \neg b(X)$
$c(t) \leftarrow \neg d_{r_4}$
$e(s) \leftarrow \neg d_{r_5}$
$d(t) \leftarrow \neg d_{r_6}$
$d(s) \leftarrow \neg d_{r_7}$
$a_{r_1} \leftarrow$
$a_{r_2}(Y) \leftarrow e(Y)$

$a_{r_3}(X) \leftarrow$
$a_{r_4} \leftarrow$
$a_{r_5} \leftarrow$
$a_{r_6} \leftarrow$
$a_{r_7} \leftarrow$
$d_{r_1}(Z) \leftarrow d(Z), a_(r_2)(Z)$
$d_{r_1}(t) \leftarrow d(t), a_{r_4}$
$d_{r_2}(t) \leftarrow d(t), a_{r_4}$
$\leftarrow a(X), b(X), \neg d_{r_3}(X), d(X)$
$\leftarrow a(X), c(X), \neg d_{r_3}(X), d(X)$
$\leftarrow b(X), c(X), \neg d_{r_3}(X), d(X)$

6 Complexity

In this section we investigate the complexity and expressive power of our system. We assume that the reader is familiar with the basic notions and definitions in this area. For a succinct but detailed overview we refer to [4].

In [16], it was shown that ordered choice logic programming is capable of representing extended generalised logic programs (EGLP) [10] while maintaining the answer set semantics as the skeptical answer semantics of the corresponding OCLP. One can prove that the same applies for the credulous answer set semantics, using the same transformation. For EGLP it can be shown that classical negation can easily be removed with the use of a single pre-processor. Therefore, we will assume that all programs are free of classical negation.

Definition 13. *Let P be an EGLP without classical negation.. The corresponding OCLP is defined by $\langle \{C, R, N\}, C \prec R \prec N \rangle$ with*

$$\begin{aligned} N &= \{\mathrm{not}_a \leftarrow \ \mid a \in \mathcal{B}_P\} \ , \\ R &= \{a \leftarrow B, \mathrm{not}_C \in R \mid r : a \leftarrow B, \neg C \in P\} \cup \\ &\quad \{\mathrm{not}_a \leftarrow B, \mathrm{not}_C \in R \mid r : \neg a \leftarrow B, \neg C \in P\} \cup \\ &\quad \{ \ \leftarrow B, \mathrm{not}_C \in R \mid r : \ \leftarrow B, \neg C \in P\} \ , \\ C &= \{a \oplus \mathrm{not}_a \leftarrow a \mid a \in \mathcal{B}_P\} \ , \end{aligned}$$

where, for $a \in \mathcal{B}_P$, not_a is a fresh atom[11] *representing $\neg a$.*

Intuitively, the choice rules in C force a choice between a and $\neg a$ while the rules in N encode "negation by default".

Theorem 5. *Let P be an EGLP without classical negation Then, $M \subseteq \mathcal{B}_P$ is an answer set of P iff S is a skeptical/credulous answer set of P_L with $S = M \cup \mathrm{not}_{(\mathcal{B}_P \setminus M)}$.*

Combining this result with the transformations from the previous section, we obtain a mechanism to go, at any time, from a logic program to an ordered choice logic programs and back without changing the semantics of our programs. All three mappings involved in this process can be executed in polynomial time, which implies that semi-negative and ordered choice logic programming have the same complexity and expressive power as far as their (skeptical/credulous) answer set semantics is concerned. A good overview of complexity and expressive power of logic programming can be found in [14].

Theorem 6. *Let P be an ordered choice logic program:*

- *Checking if an interpretation M is a skeptical/credulous answer set of P is **P**-complete.*

[10] Extended generalised logic programs allow both types of negation (classical and as failure) to appear both in the head and body of a rule. negation as failure in the head of rule. See [25] for more information details.

[11] For a set $X \in \mathcal{B}_P$, $\mathrm{not}_X = \{\mathrm{not}_a \mid a \in X\}$.

- *Deciding if P has a skeptical/credulous answer set is **NP**-complete.*
- *Ordered choice logic programming under the skeptical/credulous answer set semantics is co-**NP**-complete.*
- *If we allow function symbols with non-zero arity, the complexity of OCLP becomes: $\mathbf{\Pi}^1_1$-complete.*
- *Full OCLP under the skeptical/credulous semantics expresses or captures $\mathbf{\Pi}^1_1$.*

7 Relationship to Other Approaches

Various logic (programming) formalisms have been introduced to deal with the notions of preference, order and updates. Ordered choice logic programming uses the intuition of defeating from ordered logic programming (OLP) [22, 24] to select the most favourable alternative of a decision. In fact, every ordered logic program can be transformed into a OCLP such that the answer set semantics reflects the credulous semantics of the OCLP.

Dynamic preference in extended logic programs was introduced in [8] in order to obtain a better suited well-founded semantics. Although preferences are called dynamic they are not dynamic in our sense. Instead of defining a preference relation on subsets of rules, preferences are incorporated as rules in the program. Moreover, a stability criterion may come into play to overrule preference information. Another important difference with our approach is the notion of alternatives, as the corresponding notion in [8] is statically defined.

A totally different approach is proposed in [28]. where preferences are defined between atoms. Given these preferences, one can combine them to obtain preferences for sets of atoms. Defining models in the usual way, the preferences are then used to filter out the less preferred models.

[11] proposes disjunctive ordered logic programs which are similar to ordered logic programs [22] where disjunctive rules are permitted. In [10], preference in extended disjunctive logic programming is considered. As far as overriding is concerned the technique corresponds to a skeptical version of the OCLP semantics ([16]), but alternatives are fixed as an atom and its (classical) negation.

To reason about updates of generalised logic programs, extended logic programs without classical negation, [2] introduces dynamic logic programs. A stable model of such a dynamic logic program is a stable model of the generalised program obtained by removing the rejected rules. The definition of a rejected rule corresponds to our definition of a defeated rule when a and $\neg a$ are considered alternatives. It was shown in [2], that the stable model semantics and the answer set semantics coincide for generalised logic programs. In Theorem 5 we have demonstrated that extended logic programs without classical negation can be represented as ordered choice logic programs such that the answer set semantics of the extended logic program can be obtained as the answer set semantics of the OCLP. Because rejecting rules corresponds to defeating rules, it is not hard to see that, with some minor changes, Definition 13 can be used to retrieve the stable models of the dynamic logic program as the stable models of the corresponding OCLP. The only things we need to do are to replace the component R by the P_is of the dynamic logic program $\bigoplus\{P_i : i \in S\}$, replace every occurrence of $\neg a$ by not_a and add $a \oplus \mathrm{not}_a \leftarrow \mathrm{not}_a$ to C for each $a \in \mathcal{B}_P$.

A similar system is proposed in [19], where sequences are based on extended logic programs, and defeat is restricted to rules with opposing heads. The semantics is obtained by mapping to a single extended logic program containing expanded rules such that defeated rules become blocked in the interpretation of the "flattened" program.

A slightly different version of Definition 13 can be used to map the sequences of programs of [19] to OCLPs.

In [3], preferences are added to the dynamic logic program formalism of [2]. These are used to select the most preferred stable models. [29] also proposes a formalism that uses the order among rules to induce an order on answer sets for inconsistent programs, making it unclear on how to represent decisions. Along the same line, [18] proposes logic programs with compiled preferences, where preferences may appear in any part of the rules. For the semantics, [18] maps the program to an extended logic program.

[6, 9, 7] use yet an other approach. They define preferences between the head atoms of a disjunction. The first head atom is more preferred than the second which is more preferred that the third, etc. So in a sense one can say that these atoms become alternatives. In our approach we define the preferences between the alternatives in separate rules which each conditions of their own. To our opinion this allows for a greater sense of freedom for the programmer. Furthermore, in our system, it is very easy to adapt to changes over time, as decisions can be overruled or defeated. However, one thing has definitely to be said for their approach, they can represent problems in Σ_2^P while our approach is restricted to Σ_1^P. This higher order of complexity is achieved by allowing multiple alternatives to be decided upon without overruling the decision.

8 Conclusions and Directions for Future Research

In this paper we proposed a mechanism for transforming ordered choice logic programs to semi-negative logic program while preserving, depending on the transformation, the skeptical or credulous answer set semantics. We examined the possible ways in which the proposed theoretical mappings could be made more efficient, when used on top of an answer set solver. On the more theoretical side, these transformations allowed us to study the complexity and the expressiveness of our formalism.

Previously, OCLP was used to describe and to reason about game theory ([16, 17]). It was shown that OCLP is an elegant and intuitive mechanism for representing extensive games with perfect information. The Nash and subgame perfect equilibria can easily be obtained as the credulous answer sets of the corresponding programs. To this extent, we used a special class of OCLPs. Combining these special characteristics with the mapping of OCLP to logic programs, we can create a game-theory tailored front-end to answer set solvers.

In [17], we proposed a multi-agent system where the knowledge and beliefs of the agents is modelled by an OCLP. The agents communicate with each other by sending skeptical/credulous answer sets. The notion of evolutionary fixpoint shows how the various agents reason in order to come to their final conclusions. Having an implementation for OCLP would allow us to implement multi-agent systems and experiment with them in various domains. One possibility would be incorporating this technology into Carel ([30]), a multi-agent system for organ and tissue exchange.

In the same domain of multi-agent systems it would be interesting to find out if, with this technique, we could include preferences in the DALI-system ([13]). Future research will also focus on using OCLP as a tool in multi-agent institutions [27] to enforce norms and regulations [21] between participating agents.

References

1. *Logic in Artificial Intelligence*, volume 1919 of *Lecture Notes in Artificial Intelligence*, Cosenza, Italy, September 2002. Springer Verlag.
2. José Júlio Alferes, Leite J. A., Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic logic programming. In Cohn et al. [12], pages 98–111.
3. José Júlio Alferes and Luís Moniz Pereira. Updates plus preferences. In *European Workshop, JELIA 2000*, volume 1919 of *Lecture Notes in Artificial Intelligence*, pages 345–360, Malaga, Spain, September–October 2000. Springer Verslag.
4. Chitta Baral. *Knowledge Representation, Reasoning and Declarative Problem Solving*. Cambridge Press, 2003.
5. Martin Brain and Marina De Vos. Implementing OCLP as a front-end for Answer Set Solvers: From Theory to Practice. In *ASP03: Answer Set Programming: Advances in Theory and Implementation*. Ceur-WS, 2003. online CEUR-WS.org/Vol-78/asp03-final-brain.ps.
6. G. Brewka, Benferhat, and D. Le Berre. Qualitive Choice Logic. In *Principles of Knowledge Representation and Reasoning. KR-02*. Morgan Kaufmann, 2002.
7. G. Brewka, I Niemelä, and M Truszczynski. Answer set programming. In *International Joint Conference on Artificial Intelligence (JCAI 2003)*. Morgan Kaufmann, 2003.
8. Gerhard Brewka. Well-Founded Semantics for Extended Logic Programs with Dynamic Preferences. *Journal of Artificial Intelligence Research*, 4:19–36, 1996.
9. Gerhard Brewka, Ilkka Niemelä, and Tommi Syrjänen. Implementing ordered disjunction using answer set solvers for normal programs. In *European Workshop, JELIA 2002* [1].
10. Francesco Buccafurri, Wolfgang Faber, and Nicola Leone. Disjunctive Logic Programs with Inheritance. In Danny De Schreye, editor, *International Conference on Logic Programming (ICLP)*, pages 79–93, Las Cruces, New Mexico, USA, 1999. The MIT Press.
11. Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Disjunctive ordered logic: Semantics and expressiveness. In Cohn et al. [12], pages 418–431.
12. Anthony G. Cohn, Lenhard K. Schubert, and Stuart C. Shapiro, editors. *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning*, Trento, June 1998. Morgan Kaufmann.
13. Stefania Costantini and Arianna Tocchio. A Logic programming Language for Multi-Agent Systems. In *Logics in Artificial Intelligence, Jelia2002*, number 2424 in Lecture Notes in Artificial Intelligence, 2002.
14. Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and expressive power of logic programming. *ACM Computing Surveys (CSUR)*, 33(3):374–425, 2001.
15. Marina De Vos and Dirk Vermeir. On the Role of Negation in Choice Logic Programs. In Michael Gelfond, Nicola Leone, and Gerald Pfeifer, editors, *Logic Programming and Non-Monotonic Reasoning Conference (LPNMR'99)*, volume 1730 of *Lecture Notes in Artificial Intelligence*, pages 236–246, El Paso, Texas, USA, 1999. Springer Verslag.
16. Marina De Vos and Dirk Vermeir. Dynamic Decision Making in Logic Programming and Game Theory. In *AI2002: Advances in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, pages 36–47. Springer, December 2002.

17. Marina De Vos and Dirk Vermeir. Logic Programming Agents Playing Games. In *Research and Development in Intelligent Systems XIX (ES2002)*, BCS Conference Series, pages 323–336. Springer, December 2002.
18. J. Delgrande, T. Schaub, and H. Tompits. Logic programs with compiled preferences. In W. Horn, editor, *European Conference on Artificial Intelligence*, pages 392–398, 2000.
19. Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. On Properties of update Sequences Based on Causal Rejection. *Theory and Practice of Logic Programming*, 2(6), November 2002.
20. Thomas Eiter, Nicola Leone, Cristinel Mateis, Gerald Pfeifer, and Francesco Scarcello. The KR system `dlv`: Progress report, comparisons and benchmarks. In Anthony G. Cohn, Lenhart Schubert, and Stuart C. Shapiro, editors, *KR'98: Principles of Knowledge Representation and Reasoning*, pages 406–417. Morgan Kaufmann, San Francisco, California, 1998.
21. Marc Esteva, Julian Padget, and Carles Sierra. Formalizing a language for institutions and norms. In Jean-Jules Meyer and Milinde Tambe, editors, *Intelligent Agents VIII*, volume 2333 of *Lecture Notes in Artificial Intelligence*, pages 348–366. Springer Verlag, 2001. ISBN 3-540-43858-0.
22. D. Gabbay, E. Laenens, and D. Vermeir. Credulous vs. Sceptical Semantics for Ordered Logic Programs. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning*, pages 208–217, Cambridge, Mass, 1991. Morgan Kaufmann.
23. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of fifth logic programming symposium*, pages 1070–1080. MIT PRESS, 1988.
24. Els Laenens and Dirk Vermeir. A Universal Fixpoint Semantics for Ordered Logic. *Computers and Artificial Intelligence*, 19(3), 2000.
25. Vladimir Lifschitz. Answer set programming and plan generation. *Journal of Artificial Intelligence*, 138(1-2):39–54, 2002.
26. I. Niemelä and P. Simons. Smodels: An implementation of the stable model and well-founded semantics for normal LP. In Jürgen Dix, Ulrich Furbach, and Anil Nerode, editors, *Proceedings of the 4th International Conference on Logic Programming and Nonmonotonic Reasoning*, volume 1265 of *LNAI*, pages 420–429, Berlin, July 28–31 1997. Springer.
27. Joan-Antoní Rodríguez, Pablo Noriega, Carles Sierra, and Julian Padget. FM96.5 A Java-based Electronic Auction House. In *Proceedings of 2nd Conference on Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM'97)*, pages 207–224, London, UK, April 1997. ISBN 0-9525554-6-8.
28. Chiaki Sakama and Katsumi Inoue. Representing Priorities in Logic Programs. In Michael Maher, editor, *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, pages 82–96, Cambridge, September2–6 1996. MIT Press.
29. Davy Van Nieuwenborgh and Dirk Vermeir. Preferred answer sets for ordered logic programs. In *European Workshop, JELIA 2002* [1], pages 432–443.
30. Javier Vázquez-Salceda, Julian Padget, Ulises Cortés, Antonio López-Navidad, and Francisco Caballero. Formalizing an electronic institution for the distribution of human tissues. *Artificial Intelligence in Medicine*, 27(3):233–258, 2003. published by Elsevier.