Implementing OCLP as a front-end for Answer Set Solvers: From Theory to Practice

Martin Brain and Marina De Vos[*]

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

Abstract. Ordered Choice Logic Programming (OCLP) allows for preference-based decision-making with multiple alternatives and without the burden of 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. The semantics of the language is based on the preference between alternatives, yielding both a skeptical and a credulous approach. In this paper we discuss the theoretical basis for the implementation of an OCLP front-end for answer set solvers that can compute both semantics in an efficient manner. Both the basic algorithm and the proposed optimizations can be used in general and are not tailored towards any particular answer set solver.

1 Introduction

Examining human reasoning, we find that people often use preference, order or defaults for making decisions: "I prefer this dish", "This color 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 modeling 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. [11, 10] 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 opposite (contradictory) consequences, thus statically limiting the number of alternatives of a decision.

In [8], we proposed a formalism, 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.

[*] 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.

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 to more specific or preferred components until a conflict among alternatives arises, in which case the most specific one will be favored. 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 it is better to remain indecisive when two of your friends have an argument! To allow both types of intuitive reasoning, a credulous and skeptical semantics are introduced.

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 of negation (classic and as-failure) can easily be simulated.

In this paper, we propose a basic algorithm and optimizations for building an OCLP front-end for answer set solvers. Smodels ([12]), developed at Helsinki University of Technology, and DLV ([17]), created at the Technical University of Vienna and the University of Calabria are currently the most popular ones. An implementation build on top of Smodels can be obtained from http://www.cs.bath.ac.uk/∼mdv/oct/.

The remainder of this paper is organized as follows: we continue in Section 2 with short overview of the basic information 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 a mapping of OCLP to semi-negative logic programs allowing answer set solvers to work with OCLP. These mappings, one for each semantics, can then serve as the foundations on which we build the OCLP front-end. Apart from this theoretical/naive mapping, we propose various improvements/optimizations which allow answer set solvers to handle the transformed program more efficiently. We end this paper with a discussion on the relations to other approaches (Section 5) and directions for future research (Section 6).

2 Choice Logic Programming

Choice logic programs [7] represent decisions by interpreting the head of a rule as an exclusive choice between alternatives.

Formally, a *Choice Logic Program* [7], 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, 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*[1] 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 modeling choice, we have that r is *applied* when r is applicable and[2] $|H_r \cap I| = 1$. 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 organized in a strict partial order[3] 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 $\mathcal{C}$.*

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 labeled by the component from which they originate and we use $c(r)$ to denote the component*[4] *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 $|H_r \cap I| = 1$.*

[1] 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.

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

[3] 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 $b \in A$ with $a \neq b$.

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

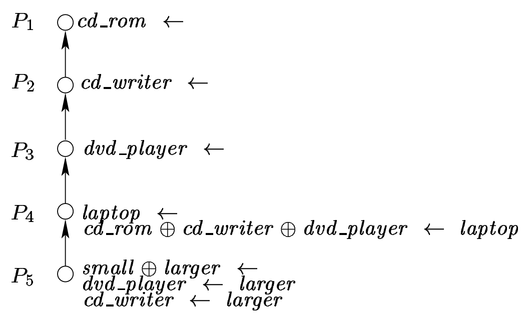
P_1
$cd\_rom \leftarrow$
P_2
$cd\_writer \leftarrow$
P_3
$dvd\_player \leftarrow$
P_4
$laptop \leftarrow$
$cd\_rom \oplus cd\_writer \oplus dvd\_player \leftarrow laptop$
P_5
$small \oplus larger \leftarrow$
$dvd\_player \leftarrow larger$
$cd\_writer \leftarrow larger$

Fig. 1. The Configuration OCLP of Example 2

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 a something similar as long as we also take the preference order into account. We want to make sure that we 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*[5]*:* $\Omega^I_C(a) = \{b \mid \exists r \in P^\star \cdot c(r) \preccurlyeq C \ \wedge \ B_r \subseteq I \ \wedge \ a, b \in H_r \text{ with } a \neq b\}$.

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

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

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 and thus no decision is made by that trial. 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 $a \leftarrow$ is not defeated as the only alternative of a, i.e. b, is not brought forward in a more preferred component.

Just as for the skeptical semantics we need to define an appropriate defeating strategy. 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.

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. $I \,\wedge\, H_{r'} \subseteq \Omega^I_{c(r)}(a)$.*

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: $\{a\}$ and $\{b\}$.

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 [14] and reduct ([16]) 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_c^M), 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*[6] *for P^M (resp. P_c^M).*

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[16]. The same applies for credulous answer sets of ordered choice logic programs, as demonstrated by the following example.

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 w.r.t. M_1 has become c-defeated w.r.t. M_2, 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 ([12]) and DLV ([17]).

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 transformation can be used for constructing an OCLP front-end. After introducing a naive mapping, we propose a number of general, not answer solver dependent, optimizations to improve efficiency of this algorithm.

[6] The definition in [8] states a stable model, but since both are identical for CLP, we have opted in this paper to use the notion of minimal model instead.

4.1 Skeptical Mapping

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 $\mathcal{A}^P_C(a)$, is defined as $\mathcal{A}^P_C(a) = \{(b, B_r) \mid \exists r \in P \cdot c(r) \preceq C, a, b \in H_r, a \neq b\}$.*

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 $\mathcal{A}^P_{P_1}(a) = \{(b, \{d\}), (c, \{d\}), (d, \{f\})\}$.

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 $\mathcal{D}^P(r_1) = \{(\{r_5\}, \{d, f\})\}$.

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

Theorem 2. *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 $P_\neg$ 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 2, 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:

$a \leftarrow \neg d_{r_1}$	$a \leftarrow f, \neg d, \neg d_{r_4}$	$a_{r_2} \leftarrow$	$\leftarrow d, \neg d_{r_3}, a, b$
$f \leftarrow \neg d_{r_2}$	$d \leftarrow f, \neg a, \neg d_{r_4}$	$a_{r_3} \leftarrow d$	$\leftarrow d, \neg d_{r_3}, a, c$
$a \leftarrow d, \neg b, \neg c, \neg d_{r_3}$	$d \leftarrow \neg c, \neg d_{r_5}$	$a_{r_4} \leftarrow f$	$\leftarrow d, \neg d_{r_3}, b, c$
$b \leftarrow d, \neg a, \neg c, \neg d_{r_3}$	$c \leftarrow \neg d, \neg d_{r_5}$	$a_{r_5} \leftarrow$	$\leftarrow f, \neg d_{r_4}, a, d$
$c \leftarrow d, \neg a, \neg b, \neg d_{r_3}$	$a_{r_1} \leftarrow$	$d_{r_1} \leftarrow a_{r_5}, d, f$	$\leftarrow \neg d_{r_5}, d, c$

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 $P_\neg$.

Theorem 3. *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. $M\}$.*

4.2 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 $\mathcal{F}^P_C(a)$, is defined as $\mathcal{F}^P_C(a) = \{(r, S) \mid \exists r \in P \cdot C \not\prec c(r), \forall b \in H_r \cdot (b, B_b) \in \mathcal{A}^P_C(a), S = \bigcup B_b\}$. 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 $(r_a, S_a) \in \mathcal{F}^P_{c(r)}(a), r_a \in R\}$.*

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

Theorem 4. *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 $\exists (R, S) \in \mathcal{F}^P_{c(r)}(a) \cdot S \subseteq I, r'$ applied w.r.t. $I, \forall r' \in R$.*

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

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

$g \leftarrow \neg d_1$	$a_1 \leftarrow g$	$d_1 \leftarrow a_2$
$p \leftarrow \neg d, \neg d_2$	$a_2 \leftarrow p, \neg d$	$d_2 \leftarrow a_3, a_4$
$d \leftarrow \neg p, \neg d_2$	$a_2 \leftarrow d, \neg p$	$d_3 \leftarrow a_2, a_4$
$g \leftarrow \neg p, \neg d_3$	$a_3 \leftarrow g, \neg p$	$d_4 \leftarrow a_2, a_3$
$p \leftarrow \neg g, \neg d_3$	$a_3 \leftarrow p, \neg g$	
$g \leftarrow \neg d, \neg d_4$	$a_4 \leftarrow g, \neg d$	
$d \leftarrow \neg g, \neg d_4$	$a_4 \leftarrow d, \neg g$	

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 5. *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. $M\}$.*

4.3 Implementing an OCLP Front End to Smodels

To demonstrate the theoretical mapping and to serve as a basis for future experimentation and research a simple language was developed to allow OCLP to be processed by computer. A compiler[7] was created to parse the input language and interface into the Smodels([12]) API which was then used to compute the answer set. The compiler *OCT* is available under the GPL ("open source") from http://www.cs.bath.ac.uk/∼mdv/oct/.

4.4 Optimizations

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 optimizations are necessary before we have an effective algorithm for converting and solving OCLPs.

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

Optimizations is used here in the context of compiler optimizations. The output will not be 'optimal' - it will be improved. Given that all of the information required to create answer sets exists at the OCLP level it would be possible to produce an 'optimal' output - it would be answer sets of the OCLP. However an answer set solver is being used to deliberately reduce the amount of logic needed when processing OCLPs and to take advantage of the optimizations and heuristics already incorporated answer set solvers. Therefore, we shall only look for simple optimizations, based on information obtained when creating the semi-negative logic program, to reduce the numbers of rules and atoms in the output, allowing answer set solvers to produce solutions more effectively. To this extent, the wording of optimization refers to the whole process and not just the compiler.

There are two key categories of optimization. The first are changes in how an individual OCLP rule is translated. These *intra-transform* optimizations don't effect the translations of any other rules and can thus be done as the rules are being translated. The other category are *inter-transform* optimizations and these are slightly more complicated. They can remove some of the simple interactions between rules and simplify the problem. However they effect the translation of other rules and thus can't be applied immediately (removing an atom from the system completely is not much use if you've already used it) but require a separate pass.

The first intra-transform improvement that can be made is to reduce the number of times that the body of any rule is included in the output. This is done by adding an extra atom $b_r \leftarrow B_r$ for each rule and then using b_r instead of B_r for the other rules. Essentially 'factoring out' the condition that the body must apply. This is of-course only a significant saving if the rule has more than one element in the body. In the case of the skeptical mapping this can be combined with a_r.

The next improvements can be made while creating the c)-rules for every rule in the OCLP. If there are no possible future (c-)defeaters for at least one of the elements in the head of a rule then there will be no rules of the form $d_r \leftarrow C$ generated and the $\neg d_r$ can be dropped from any other rules created. Conversely if there are any rules of the form $d_r \leftarrow$ that would be generated then any rule that would contain $\neg d_r$ can be ignored as the rule is considered to be automatically defeated. At the intra-transform level rules of this sort can only be located while performing a skeptical mapping as the applicability of an arbitrary rule can be determined easily but whether it is applied or not is non trivial. This improvement implies that before any other rule for r is created, the c)-rules should be constructed.

There are several inter-transform optimizations. To add to the additional problems of using these operations, applying these can result in more rules to which they can be applied, essentially requiring looping until there are no more possible improvements that can be made. To optimize the entire system, it suffices that the compiler only uses the info it directly obtains from completing transformation. The rest can be left for the other components.

In order of application and increasing complexity:

- Propagation - Semi-negative logic programming rules of the form $a \leftarrow$ state a fact about the system so this can be used to simplify the system. All references to a can be removed from the bodies of all of the other rules in the system as it will appear in any answer set[8]. Any rule with a reference to $\neg a$ in it can be removed for similar reasons. If the atom a is a constructed atom used in the transformation (i.e. a_r or d_r for some $r \in P$) then the definition $a \leftarrow$ can be removed as well as it doesn't add anything to the final answer set (in OCLP terms).

– Removal - Atoms of the form a_r, b_r and d_r can be removed if they are only found in the heads of rules. This is because all atoms of this form are removed from the stable model solution when it is mapped back to an OCLP solution. Thus if they do not form part of a condition on another rule they don't need to be calculated. The list of which of these atoms are used can be generated while the translation of rules is being made, thus saving having to do another pass. However this is not as much of a improvement as it may first seem. In the skeptical case rules using the generated atom a_r are used as an alias for b_r or will be removed by propagation in all cases except a body size of 1 (which could also be recognized and removed via aliasing optimizations). The credulous case may however benefit from removal of some a_r atoms. While the nature of b_r and the optimizations applied to see if a rule can be defeated before generating atoms of the type d_r means these are only likely to be removed like this after other optimizations have been applied (which in turn will require and extra pass to work out which generated atoms are used).
– Aliasing - Each rule of the form $a \leftarrow b$ essentially makes a an alias for b. Thus any rule who's body contains both a and b can safely remove a. The other forms of aliasing and their consequences depend on what type of atom a is.

  • $a \leftarrow b$ (a is an atom from the original OCLP) if a appears in the head of only one rule then all occurrences of a in the bodies of other rules can be replaced with b.
  • $a_r \leftarrow b$ If there is only one rule with a_r in the head (as will happen with a skeptical and some credulous mappings) all occurrences of a_r can be replaced and the rule then removed completely as it adds nothing to the final answer set.
  • $d_r \leftarrow b$ There may well be more than one rule giving conditions for d_r to be true, but if this is not the case then all occurrences of d_r can be replaced with b and this rule removed.
  • $b_r \leftarrow b$ Although these shouldn't be generated directly it is possible they will arise through propagation. Again there will only be one rule with b_r in the head so it can be replaced in all bodies and then removed.

  When replacing the atoms of a union with the body of the rule is needed as it is possible that the rule already contained b (of-course this can then create rules of the form $a \leftarrow b$ which can then be optimized again, however it will not create rules that can be reduced via propagation). This stage can also be used to eliminate any duplicate rules which might arise as the result of the mapping. This is also the ideal place to remove useless rules like $b \leftarrow b, B$ and $a \leftarrow b, \neg b, B$ which have nothing to add to the semantics of our program.

[8] Care must be taken to note the case of removing the last atom from the body of a constraint in this fashion as rather than reducing the complexity of the problem it signifies that the program has no answers. E.g. A semi-negative logic program containing the rules b and $\leftarrow b$ will have no solutions.

– Factoring - For every pair of rules $a \leftarrow B$ and $c \leftarrow D$ if $|B \cap D| \geq 2$ the common elements can be 'factored out'. For example:
  $a \leftarrow b_1, b_2, b_3, b$
  $c \leftarrow b_1, b_2, b_3, b$
  Gives
  $a \leftarrow e, b$
  $c \leftarrow e, b$
  $e \leftarrow b_1, b_2, b_3$

  This transformation should produce more compact (in terms of the total number of atoms) rules when dealing with OCLPs that have more complex possible future (c-)defeaters. However the ordering which pairs of rules should be considered and the implications of common expressions shared between more than two blocks make the application order very difficult to calculate quickly. It is possible to apply this to a smaller degree to c)-rules during the transformation process if the possible future (c-)defeaters of each atom are handled. Constructing the possible future (c-)defeaters on a rule by rule basis doesn't give fine enough 'granularity' to apply this kind of optimization.

5 Relationship to Other Approaches

Our formalism shows similarities with ordered logic programming [13, 15, 5], where the latter supports disjunction (in the head), which also provides a skeptical and a credulous approach. However, defeat is restricted to rules with contradictory heads, making it difficult to represent more complex decisions. In [4], preference in extended disjunctive logic programming is considered. As far as overriding is concerned, the technique corresponds rather well with our skeptical defeating, but, again, alternatives are limited to an atom and its (classical) negation.

To reason about updates of generalized logic programs, extended logic programs without classical negation, [1] introduces dynamic logic programs. A stable model of such a dynamic logic program is a stable model of the generalized 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. A similar system is proposed in [11], 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.

In [8], a mapping from extended logic programs to OCLP was presented. A very similar mapping allows us to map both dynamic logic programs as sequences of extended logic program to OCLP.

[2] added a system of preference to the dynamic logic programs of [1]. This preference is used to select the most preferred stable models. A similar mechanism is also used by [3] to obtain preferred answer sets: preferences are used to filter out unwanted candidate models, they are not used during model creation as is the case for OCLP.

[18] 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, [10] proposes logic programs with compiled preferences, where preferences may appear in any part of the rules. For the semantics, [10] maps the program to an extended logic program.

6 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. Having such a transformation allows an implementation of OCLP on top of answer set solvers like Smodels ([12]), and DLV ([17]). The mapping and the optimizations we proposed are very general and not directed towards any particular answer set solver. For the future we plan to experiment with the special construct provided by the different implementations. It would be interesting to find out whether incorporating them into the output of our compiler would improve the efficiency of the entire system. For this we think for example at the disjunctive rules provided by DLV. They would reduce rules of type a), while the special choice construct of Smodels would reduce both rules of type a) and d). Although this construct would reduce the number of rules in the output of our compiler, this does not automatically make the code more efficient, this depends on what these systems do with them. In case they have a special mechanism for handling them this would indeed mean a gain in effectiveness. If, however they translate everything back to standard rules, using them would only have a negative effect. It also introduces additional complications into the mapping and inter-transform optimizations and may limit their effectiveness.

Previously, OCLP was used to describe and to reason about game theory ([8, 9]). To this extend, we used a special class of OCLPs. Each atom appears exactly once in a choice rule and none of the choice rules can be defeated. Combining this knowledge with the mapping of OCLP to logic programs, we can create a game theory tailored front-end to answer set solvers.

In [9], we proposed a multi-agent system were the knowledge and beliefs of the agents is modeled by an OCLP. The agents communicate by sending answer sets, skeptical or credulous, to each other. The notion of evolutionary fixpoint shows how the various agents reasoned in order to come to their final conclusions. Having an implementation for OCLP would allow us to implement multi-agent systems and run experiments in various domains. One possibility would be incorporate this knowledge into Carrel ([19]), a multi-agent system for organ and tissue exchange.

References

1. José Júlio Alferes, Leite J. A., Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic logic programming. In Cohn et al. [6], pages 98–111.
2. 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.

3. Gerhard Brewka and Thomas Eiter. Preferred answer sets for extended logic programs. *Artificial Intelligence*, 109(1-2):297–356, April 1999.
4. 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.
5. Francesco Buccafurri, Nicola Leone, and Pasquale Rullo. Disjunctive ordered logic: Semantics and expressiveness. In Cohn et al. [6], pages 418–431.
6. 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.
7. 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.
8. 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.
9. 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.
10. J. Delgrande, T. Schaub, and H. Tompits. Logic programs with compiled preferences. In W. Horn, editor, *European Conference on Artficial Intelligence*, pages 392–398, 2000.
11. 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.
12. 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.
13. 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.
14. 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.
15. Els Laenens and Dirk Vermeir. A Universal Fixpoint Semantics for Ordered Logic. *Computers and Artificial Intelligence*, 19(3), 2000.
16. Vladimir Lifschitz. Answer set programming and plan generation. *Journal of Artificial Intelligence*, 138(1-2):39–54, 2002.
17. 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 Programing and Nonmonotonic Reasoning*, volume 1265 of *LNAI*, pages 420–429, Berlin, July 28–31 1997. Springer.
18. Davy Van Nieuwenborgh and Dirk Vermeir. Preferred answer sets for ordered logic programs. In *European Workshop, JELIA 2002*, volume 1919 of *Lecture Notes in Artificial Intelligence*, pages 432–443, Cosenza, Italy, September 2002. Springer Verlag.
19. 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.