Logic Programs with Context-Dependent Preferences*

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

Abstract

The paper describes an extension of well-founded semantics for logic programs with two types of negation. In this extension information about preferences between rules can be expressed in the logical language and derived dynamically. This is achieved by using a reserved predicate symbol and a naming technique. Conflicts among rules are resolved whenever possible on the basis of derived preference information. A legal reasoning example illustrates the usefulness of the approach.

1 Introduction: Why Dynamic Preferences are Needed

Preferences among defaults play a crucial role in nonmonotonic reasoning. One source of preferences that has been studied intensively is specificity [12, 18, 19]. In case of a conflict between defaults we tend to prefer the more specific one since this default provides more reliable information. Also in other application domains, like model based diagnosis or configuration, preferences play a fundamental role.

The relevance of preferences is well-recognized in nonmonotonic reasoning, and prioritized versions for most of the nonmonotonic logics have been proposed, e.g., prioritized circumscription [9], hierarchic autoepistemic logic [8], prioritized default logic [3]. In these approaches preferences are handled in an "external" manner in the following sense: some ordering among defaults is used to control the generation of the nonmonotonic conclusions. For instance, in the case of prioritized default logic this information is used to control the generation of extensions. However, the preference information itself is not expressed in the logical language. This means that this kind of information has to be fully pre-specified, there is no way of reasoning about (as opposed to reasoning with) preferences. This is in stark contrast to the way people reason and argue with each other. In legal argumentations, for instance, preferences are context-dependent, and the assessment of the preferences among involved conflicting laws is a crucial (if not the most crucial) part of the reasoning.

*The full version of this paper was published in Journal of Artificial Intelligence Research, 4 (1996), 19-36.

What we would like to have, therefore, is an approach that allows us to represent preference information *in* the language and derive such information dynamically[1]. In a recent paper [4] the author has described a variant of normal default logic in which reasoning about preferences is possible. Although the version of default logic presented in this earlier paper produces reasonable results in most cases, this approach has several drawbacks:

1. The approach is computationally extremely demanding as it involves the construction of the Reiter extensions and an additional compatibility check for each extension guaranteeing that the preference information was taken into account adequately.

2. It may happen that consistent default theories, i.e., theories whose strict part is satisfiable, possess no extensions at all. This is astonishing since in that paper we only dealt with normal defaults. The non-existence of extensions is due to defeasible preference information. It is highly questionable whether such information should be able to destroy all extensions.

3. The earlier paper did not take non-normal defaults into account, it is thus not general enough to cover normal logic programs with negation as failure.

The approach presented in this paper will be based on extended logic programs with two types of negation. This means that in comparison with our earlier proposal we are more restrictive in one respect and more general in another: we are more restrictive since we do not allow arbitrary first order formulas as in normal default logic; we are more general since we admit negation as failure and hence rules which correspond to non-normal defaults in Reiter's logic. We also switch from the extension based semantics of default logic to well-founded semantics [5, 15, 10], i.e., to an inherently skeptical approach where the nonmonotonic conclusions are defined directly, not through the notion of extensions. It is well-known that well-founded semantics sometimes loses intuitively expected conclusions. This is also the case in our proposal. However, this is outweighed by a tremendous gain in efficiency: the well-founded conclusions can be computed in polynomial time.

The outline of the rest of the paper is as follows: in Section 2 we first review a definition of well-founded semantics for logic programs with two types of negation which is based on the double application of a certain anti-monotone operator. The definition extends Baral and Subrahmanian's formulation of well-founded semantics for normal logic programs [2] and was used by several authors [1, 10]. We show that this definition suffers from an unnecessary weakness and present a reformulation that leads to better results. Section 3, the main section of the paper, introduces our dynamic treatment of preferences together with several small motivating examples. We show that our conclusions are, in general, a superset of the well-founded conclusions. Section 4 illustrates the expressive power of our approach using a more realistic example from legal reasoning. Section 5 concludes.

2 Well-Founded Semantics for Extended Logic Programs

A (propositional) extended logic program consists of rules of the form

$$c \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m$$

[1]Our use of the term *dynamic* is intended to express change from context to context rather than change over time.

where the a_i, b_j and c are propositional literals, i.e., either propositional atoms or such atoms preceded by the classical negation sign. The symbol *not* denotes negation by failure (weak negation), $\neg$ denotes classical (strong) negation. For convenience we will sometimes use a rule schema to represent a set of propositional rules, namely the set of all ground instances of the schema.

Extended logic programs are very useful for knowledge representation purposes, see for instance [1] for a number of illustrative examples. Two major semantics for extended logic programs have been defined: (1) answer set semantics [6], an extension of stable model semantics, and (2) a version of well-founded semantics [15]. The second approach can be viewed as an efficient approximation of the first.

Let us first introduce answer sets. We say a rule $r \in P$ of the form above is defeated by a literal l if $l = b_i$ for some $i \in \{1, \ldots, m\}$. We say r is defeated by a set of literals X if X contains a literal that defeats r. Furthermore, we call the rule obtained by deleting weakly negated preconditions from r the monotonic counterpart of r and denote it with $Mon(r)$. We also apply Mon to sets of rules with the obvious meaning.

Definition 1 *Let P be a logic program, X a set of literals. The X-reduct of P, denoted P^X, is the program obtained from P by*

- *deleting each rule defeated by X, and*
- *replacing each remaining rule r with its monotonic counterpart $Mon(r)$.*

Definition 2 *Let R be a set of rules without negation as failure. $Cn(R)$ denotes the smallest set of literals that is*

1. *closed under R, and*
2. *logically closed, i.e., either consistent or equal to the set of all literals.*

Definition 3 *Let P be a logic program, X a set of literals. Define an operator γ_P as follows:*

$$\gamma_P(X) = Cn(P^X)$$

X is an answer set of P iff $X = \gamma_P(X)$.

A literal l is a consequence of a program P under answer set semantics, denoted $l \in Ans(P)$, iff l is contained in all answer sets of P.

The second major semantics for extended logic programs, well-founded semantics, is an inherently skeptical semantics that refrains from drawing conclusions whenever there is a potential conflict. The original formulation of well-founded semantics for general logic programs in [5] is based on a certain partial model. Przymusinski reconstructed this definition in 3-valued logic [14]. The formulation using an anti-monotone operator was first given in [2] for general logic programs together with a corresponding definition for default logic. The straightforward extension of this formulation (respectively, the restriction of the default logic definition) to extended logic programs that will be introduced now was used by several authors, e.g. [1, 10].[2] Note that in this paper we will only consider the literals that are true in the corresponding 3-valued semantics.

[2]Pereira and Alferes [11] later introduced a version of well-founded semantics that adheres to the so-called coherence principle which requires that strong negation implies weak negation. We will show later in Sect. 3 how the coherence principle can be introduced in our approach.

Like answer set semantics the well-founded semantics for extended logic programs is based on the operator γ_P However, the operator is used in a totally different way. Since γ_P is anti-monotone the function $\Gamma_P = (\gamma_P)^2$ is monotone. According to the famous Knaster-Tarski theorem [17] every monotone operator has a least fixpoint. The set of well-founded conclusions of a program P, denoted $WFS(P)$, is defined to be this least fixpoint of Γ_P. The fixpoint can be approached from below by iterating Γ_P on the empty set. In case P is finite this iteration is guaranteed to actually reach the fixpoint.

The intuition behind this use of the operator is as follows: whenever γ_P is applied to a set of literals X known to be true it produces the set of all literals that are still potentially derivable. Applying it to such a set of potentially derivable literals it produces a set of literals known to be true, often larger than the original set X. Starting with the empty set and iterating until the fixpoint is reached thus produces a set of true literals. It can be shown that every well-founded conclusion is a conclusion under the answer set semantics. Well-founded semantics can thus be viewed as an approximation of answer set semantics.

Unfortunately it turns out that for many programs the set of well-founded conclusions is extremely small and provides a very poor approximation of answer set semantics. Consider the following program P_0 which has also been discussed in [1]:

1) $b \leftarrow not \neg b$
2) $a \leftarrow not \neg a$
3) $\neg a \leftarrow not\ a$

The set of well-founded conclusions is empty since $\gamma_{P_0}(\emptyset)$ equals Lit, the set of all literals, and the Lit-reduct of P_0 contains no rule at all. This is surprising since, intuitively, the conflict between 2) and 3) has nothing to do with $\neg b$ and b.

This problem arises whenever the following conditions hold:

1. a complementary pair of literals is provable from the monotonic counterparts of the rules of a program P, and

2. there is at least one proof for each of the complementary literals whose rules are not defeated by $Cn(P')$, where P' consists of the "strict" rules in P, i.e., those without negation as failure.

In this case well-founded semantics concludes l iff $l \in Cn(P')$. It should be obvious that such a situation is not just a rare limiting case. To the contrary, it can be expected that many commonsense knowledge bases will give rise to such undesired behaviour.

We want to show that a minor reformulation of the fixpoint operator can overcome this intolerable weakness and leads to much better results. Consider the following operator

$$\gamma_P^\star(X) = Cl(P^X)$$

where $Cl(R)$ denotes the minimal set of literals closed under the (classical) rules R. $Cl(R)$ is thus like $Cn(R)$ without the requirement of logical closedness. Now define

$$\Gamma_P^\star(X) = \gamma_P(\gamma_P^\star(X))$$

Again we iterate on the empty set to obtain the well-founded conclusions of a program P which we will denote $WFS^\star(P)$.

Consider the effects of this modification on our example P_0. $\gamma^{\star}_{P_0}(X) = \{a, \neg a, b\}$. Rule 1) is contained in the $\{a, \neg a, b\}$-reduct of P_0 and thus $\Gamma^{\star}_{P_0}(\emptyset) = \{b\}$. Since b is also the only literal contained in all answer sets of P_1 our approximation actually coincides with answer set semantics in this case.

It is not difficult to see that the new monotone operator is equivalent to the original one whenever P does not contain negation as failure. In this case the X-reduct of P, for arbitrary X, is equivalent to P and for this reason it does not make any difference whether to use γ_P or $\gamma^{\star}_P$ as the operator to be applied first in the definition of Γ_P. The same is obviously true for programs without classical negation: for such programs Cn can never produce complementary pairs of literals and for this reason the logical closedness condition is obsolete.

In the general case the new operator produces more conclusions than the original one:

Proposition 1 *Let P be an extended logic program. For an arbitrary set of literals X we have*

$$\Gamma_P(X) \subseteq \Gamma^{\star}_P(X).$$

It remains to be shown that the new operator produces no unwanted results, i.e., that our new semantics can still be viewed as an approximation of answer set semantics.

Proposition 2 *Let P be an extended logic program. Let $Ans(P)$ be the set of literals contained in all answer sets of P. $WFS^{\star}$ is correct wrt answer set semantics, i.e., $WFS^{\star}(P) \subseteq Ans(P)$.*

For the rest of the paper a minor reformulation turns out to be convenient. Instead of using the monotonic counterparts of undefeated rules we will work with the original rules and extend the definitions of the two operators Cn and Cl accordingly, requiring that weakly negated preconditions be neglected, that is, we extend Cn and Cl to arbitrary sets of rules P with weak negation by defining $Cn(P) = Cn(Mon(P))$ and $Cl(P) = Cl(Mon(P))$. We can now equivalently characterize γ_P and $\gamma^{\star}_P$ by the equations

$$\gamma_P(X) = Cn(P_X)$$

$$\gamma^{\star}_P(X) = Cl(P_X)$$

where P_X denotes the set of rules not defeated by X.

Before we turn to the treatment of preferences we give an alternative characterization of $\Gamma^{\star}_P$ based on the following notion:

Definition 4 *Let P be a logic program, X a set of literals. A rule r is X-safe wrt P ($r \in SAFE_X(P)$) if r is not defeated by $\gamma^{\star}_P(X)$ or, equivalently, if $r \in P_{\gamma^{\star}_P(X)}$.*

With this new notion we can obviously characterize $\Gamma^{\star}_P$ as follows:

$$\Gamma^{\star}_P(X) = Cn(P^{\gamma^{\star}_P(X)}) = Cn(P_{\gamma^{\star}_P(X)}) = Cn(SAFE_X(P))$$

It is this last formulation that we will modify later. More precisely, the notion of X-safeness will be weakened to handle preferences adequately.

3 Adding Preferences

In order to handle preferences we need to be able to express preference information explicitly. Since we want to do this *in* the logical language we have to extend the language. We do this in two respects:

1. we use a set of rule names N together with a naming function *name* to be able to refer to particular rules,

2. we use a special (infix) symbol $\prec$ that can take rule names as arguments to represent preferences among rules.

Intuitively, $n_1 \prec n_2$ where n_1 and n_2 are rule names means the rule with name n_1 is preferred over the rule with name n_2.[3]

A prioritized logic program is a pair $(R, name)$ where R is a set of rules and *name* a naming function. To make sure that the symbol $\prec$ has its intended meaning, i.e., represents a transitive and anti-symmetric relation, we assume that R contains all ground instances of the schemata

$$N_1 \prec N_3 \leftarrow N_1 \prec N_2, N_2 \prec N_3$$

and

$$\neg(N_2 \prec N_1) \leftarrow N_1 \prec N_2$$

where N_i are parameters for names. Note that in our examples we won't mention these rules explicitly.

The function *name* is a partial injective naming function that assigns a name $n \in N$ to some of the rules in R. Note that not all rules do necessarily have a name. The reason is that names will only play a role in conflict resolution among defeasible rules, i.e., rules with weakly negated preconditions. For this reason names for strict rules, i.e., rules in which the symbol *not* does not appear, won't be needed. A technical advantage of leaving some rules unnamed is that the use of rule schemata with parameters for rule names does not necessarily make programs infinite. If we would require names for all rules we would have to use a parametrized name for each schema and thus end up with an infinite set N of names.

In our examples we assume that N is given implicitly. We also define the function *name* implicitly. We write:

$$n_i : c \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m$$

to express that $name(c \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m) = n_i$.

For convenience we will simply speak of programs instead of prioritized logic programs whenever this does not lead to misunderstandings.

Before introducing new definitions we would like to point out how we want the new explicit preference information to be used. Our approach follows two principles:

1. we want to extend well-founded semantics, i.e. we want that every $WFS^\star$-conclusion remains a conclusion in the prioritized approach,

2. we want to use preferences to solve conflicts whenever this is possible without violating principle 1.

[3] Note that for historical reasons we follow the convention that the minimal rules are the preferred ones.

Let us first explain what we mean by conflict here. Rules may be conflicting in several ways. In the simplest case two rules may have complementary literals in their heads. We call this a type-I conflict. Conflicts of this type may render the set of well-founded conclusions inconsistent, but do not necessarily do so. If, for instance, a precondition of one of the rules is not derivable or a rule is defeated the conflict is implicitly resolved. In that case the preference information will simply be neglected. Consider the following program P_1:

$$n_1 : b \leftarrow not\ c$$
$$n_2 : \neg b \leftarrow not\ b$$
$$n_3 : n_2 \prec n_1$$

There is a type-I conflict between n_1 and n_2. Although the explicit preference information gives precedence to n_2 we want to apply n_1 here to comply with the first of our two principles. Technically, this means that we can apply a preferred rule r only if we are sure that r's application actually leads to a situation where literals defeating r can no longer be derived.

The following two rules exhibit a different type of conflict:

$$a \leftarrow not\ b$$
$$b \leftarrow not\ a$$

The heads of these rules are not complementary. However, the application of one rule defeats the other and vice versa. We call this a direct type-II conflict. Of course, in the general case the defeat of the conflicting rule may be indirect, i.e. based on the existence of additional rules. We say r_1 and r_2 are type-II conflicting wrt. a set of rules R iff

1. $Cl(R)$ neither defeats r_1 nor r_2,

2. $Cl(R + r_1)$ defeats r_2, and

3. $Cl(R + r_2)$ defeats r_1

Here $R + r$ abbreviates $R \cup \{r\}$. A direct type-II conflict is thus a type-II conflict wrt. the empty set of rules. The rule sets R that have to be taken into account in our well-founded semantics based approach are subsets of the rules which are undefeated by the set of literals known to be true. Note that the two types of conflict are not disjoint, i.e. two rules may be in conflict of both type-I and type-II. Consider the following program P_2, a slight modification of P_1:

$$n_1 : b \leftarrow not\ c, not \neg b$$
$$n_2 : \neg b \leftarrow not\ b$$
$$n_3 : n_2 \prec n_1$$

Now we have a type-II conflict between n_1 and n_2 (more precisely, a direct type-II and a type-I conflict) that is not solvable by the implicit mechanisms of well-founded semantics alone. It is this kind of conflict that we try to solve by the explicit preference information. In our example n_2 will be used to derive $\neg b$. Note that now the application of n_2 defeats n_1 and there is no danger that a literal defeating n_2 might become derivable later. Generally, a type-II conflict between r_1 and r_2 (wrt. some undefeated rules of the program) will be solved in favour of the preferred rule, say r_1, only if applying r_1 excludes any further possibility of deriving an r_1-defeating literal.

Note that every type-I conflict can be turned into a direct type-II conflict by a (non-equivalent!) rerepresentation of the rules: if each conflicting rule r is replaced by its seminormal form[4] then all conflicts become type-II conflicts and are thus amenable to conflict resolution through preference information.

After this motivating discussion let us present the new definitions. Our treatment of priorities is based on a weakening of the notion of X-safeness. In Sect. 2 we considered a rule r as X-safe whenever there is no proof for a literal defeating r from the monotonic counterparts of X-undefeated rules. Now in the context of a prioritized logic program we will consider a rule r as X-safe if there is no such proof from monotonic counterparts *of a certain subset* of the X-undefeated rules. The subset to be used depends on the rule r and consists of those rules that are not "dominated" by r. Intuitively, r' is dominated by r iff r' is (1) known to be less preferred than r and (2) defeated when r is applied together with rules that already have been established to be X-safe. (2) is necessary to make sure that explicit preference information is used the right way, according to our discussion of P_1.

It is obvious that whenever there is no proof for a defeating literal from all X-undefeated rules there can be no such proof from a subset of these rules. Rules that were X-safe according to our earlier definition thus remain to be X-safe. Here are the precise definitions:

Definition 5 *Let $P = (R, name)$ be a prioritized logic program, X a set of literals, Y a set of rules, and $r \in R$. The set of rules dominated by r wrt X and Y, denoted $Dom_{X,Y}(r)$, is the set*

$$\{r' \in R \mid name(r) \prec name(r') \in X \text{ and } Cl(Y + r) \text{ defeats } r'\}$$

Note that $Dom_{X,Y}(r)$ is monotonic in both X and Y. We can now define the X-safe rules inductively:

Definition 6 *Let $P = (R, name)$ be a prioritized logic program, X a set of literals. The set of X-safe rules of P, denoted $SAFE_X^{pr}(P)$, is defined as follows: $SAFE_X^{pr}(P) = \bigcup_{i=0}^{\infty} R_i$, where*

$R_0 = \emptyset$, and for $i > 0$,
$R_i = \{r \in R \mid r \text{ not defeated by } Cl(R_X \setminus Dom_{X,R_{i-1}}(r))\}$

Note that X-safeness is obviously monotonic in X. Based on this notion we introduce a new monotonic operator Γ_P^{pr}:

Definition 7 *Let $P = (R, name)$ be a prioritized logic program, X a set of literals. The operator Γ_P^{pr} is defined as follows:*

$$\Gamma_P^{pr}(X) = Cn(SAFE_X^{pr}(P))$$

As before we define the (prioritized) well-founded conclusions of P, denoted $WFS^{pr}(P)$, as the least fixpoint of Γ_P^{pr}. If a program does not contain preference information at all, i.e., if the symbol $\prec$ does not appear in R, the new semantics coincides with $WFS^\star$ since in that case no rule can dominate another rule. In the general case, since the new definition of X-safeness is weaker than the one used earlier in Sect. 2 we may have more X-safe rules and for this reason obtain more conclusions than via $\Gamma_P^\star$. The following result is thus obvious:

[4]The seminormal form of $c \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m$ is

$$c \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m, not\ c'$$

where c' is the complement of c. The term seminormal is taken from [16].

Proposition 3 *Let $P = (R, name)$ be a prioritized logic program. For every set of literals X we have $\Gamma^{\star}_{R}(X) \subseteq \Gamma^{pr}_{P}(X)$.*

From this and the monotonicity of both operators it follows immediately that $WFS^{\star}(R) \subseteq WFS^{pr}(P)$.[5]

Well-founded semantics has sometimes been criticized for being too weak and missing intended conclusions. The proposition shows that we can strengthen the obtained results by adding adequate preference information.

As a first simple example let us consider the following program P_3:

$n_1 : b \leftarrow not\ c$
$n_2 : c \leftarrow not\ b$
$n_3 : n_2 \prec n_1$

We first apply $\Gamma^{pr}_{P_3}$ to the empty set. Besides the instances of the transisitvity and anti-symmetry schema that we implicitly assume only n_3 is in $SAFE^{pr}_{\emptyset}(P_3)$. We thus obtain

$$S_1 = \{n_2 \prec n_1, \neg(n_1 \prec n_2)\}$$

We next apply $\Gamma^{pr}_{P_3}$ to S_1. Since $n_2 \prec n_1 \in S_1$ we have $n_1 \in Dom_{S_1,\emptyset}(n_2)$. $n_2 \in SAFE^{pr}_{S_1}(P_3)$ since $Cl(P_{3_{S_1}} \setminus \{n_1\})$ does not defeat n_2 and we obtain

$$S_2 = \{n_2 \prec n_1, \neg(n_1 \prec n_2), c\}$$

Further iteration of $\Gamma^{pr}_{P_3}$ yields no new literals, i.e. S_2 is the least fixpoint. Note that c is not a conclusion under the original well-founded semantics.

We next show that the programs P_1 and P_2 discussed earlier are handled as intended. Here is P_1:

$n_1 : b \leftarrow not\ c$
$n_2 : \neg b \leftarrow not\ b$
$n_3 : n_2 \prec n_1$

Since $\gamma^{\star}_{P_1}(\emptyset)$ does not defeat n_1 this rule is safe from the beginning, that is, $n_1 \in SAFE^{pr}_{\emptyset}(P_1)$. $\Gamma^{pr}_{P}(\emptyset)$ yields $\{n_2 \prec n_1, \neg(n_1 \prec n_2), b\}$ which is also the least fixpoint. The explicit preference does not interfere with the implicit one, as intended.

The situation changes in P_2 where the first rule in P_1 is replaced by

$n_1 : b \leftarrow not\ c, not \neg b$

The new rule n_1 is not in $SAFE^{pr}_{\emptyset}(P_2)$ since it is defeated by the consequence of n_2 and n_2 is not dominated by n_1. $\Gamma^{pr}_{P_2}(\emptyset)$ yields

$$S_1 = \{n_2 \prec n_1, \neg(n_1 \prec n_2)\}$$

[5] In [11] Pereira and Alferes argue that each extension of well-founded semantics to two types of negation should satisfy what they call coherence principle: a weakly negated precondition should be considered satisfied whenever the corresponding strongly negated literal is derived. To model this principle in our approach one would have to weaken the notion of X-safeness even further. In the inductive definition, a rule r would have to be considered a member of R_i whenever for each weak precondition *not* b of r

- $b \notin Cl(R_X \setminus Dom_{X,R_{i-1}}(r))$, or
- $b' \in X$, where $b' = \neg b$ if b is an atom and a if $b = \neg a$.

Now $n_2 \in SAFE^{pr}_{S_1}(P_2)$ since n_2 dominates n_1 wrt. S_1 and the empty set of rules. We thus conclude $\neg b$ as intended. The least fixpoint is

$$S_2 = \{n_2 \prec n_1, \neg(n_1 \prec n_2), b\}$$

In [4] we used an example to illustrate the possible non-existence of extensions in our earlier approach. This example involved two normal defaults each of which had the conclusion that the other one is to be preferred. The prioritized logic programming representation of this example is the following:

$$n_1 : n_2 \prec n_1 \leftarrow not \neg(n_2 \prec n_1)$$
$$n_2 : n_1 \prec n_2 \leftarrow not \neg(n_1 \prec n_2)$$

It is straighforward to verify that the set of well-founded conclusions for this example is empty.

4 A legal reasoning example

In this section we want to show that the additional expressiveness provided by our approach actually helps representing real world problems. We will use an example first discussed by Gordon [7, p.7]. We somewhat simplified it for our purposes. The same example was also used in [4] to illustrate the approach presented there.

Assume a person wants to find out if her security interest in a certain ship is perfected. She currently has possession of the ship. According to the Uniform Commercial Code (UCC, §9-305) a security interest in goods may be perfected by taking possession of the collateral. However, there is a federal law called the Ship Mortgage Act (SMA) according to which a security interest in a ship may only be perfected by filing a financing statement. Such a statement has not been filed. Now the question is whether the UCC or the SMA takes precedence in this case. There are two known legal principles for resolving conflicts of this kind. The principle of *Lex Posterior* gives precedence to newer laws. In our case the UCC is newer than the SMA. On the other hand, the principle of *Lex Superior* gives precedence to laws supported by the higher authority. In our case the SMA has higher authority since it is federal law.

The available information can nicely be represented in our approach. To make the example somewhat shorter we use the notation

$$c \Leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m$$

as an abbreviation for the rule

$$c \leftarrow a_1, \ldots, a_n, not\ b_1, \ldots, not\ b_m, not\ c'$$

where c' is the complement of c, i.e. $\neg c$ if c is an atom and a if $c = \neg a$. Such rules thus correspond to semi-normal or, if $m = 0$, normal defaults in Reiter's default logic [16].

We use the ground instances of the following named rules to represent the relevant article of the UCC, the SMA, Lex Posterior (LP), and Lex Superior (LS). The symbols d_1 and d_2 are parameters for rule names:

$UCC : perfected \Leftarrow possession$
$SMA : \neg perfected \Leftarrow ship, \neg fin\text{-}statement$
$LP(d_1, d_2) : d_1 \prec d_2 \Leftarrow more\text{-}recent(d_1, d_2)$
$LS(d_1, d_2) : d_1 \prec d_2 \Leftarrow fed\text{-}law(d_1), state\text{-}law(d_2)$

The following facts are known about the case and are represented as rules without body (and without name):

$possession$
$ship$
$\neg fin\text{-}statement$
$more\text{-}recent(UCC, SMA)$
$fed\text{-}law(SMA)$
$state\text{-}law(UCC)$

Let's call the above set of literals H. Iterated application of Γ_P^{pr} yields the following sequence of literal sets (in each case $S_i = (\Gamma_P^{pr})^i(\emptyset)$):

$$\begin{aligned} S_1 &= H \\ S_2 &= S_1 \end{aligned}$$

The iteration produces no new results besides the facts already contained in the program. The reason is that UCC and SMA block each other, and that no preference information is produced since also the relevant instances of Lex Posterior and Lex Superior block each other. The situation changes if we add information telling us how conflicts between the latter two are to be resolved. Assume we add the following information:[6]

$$LS(SMA, UCC) \prec LP(UCC, SMA)$$

Now we obtain the following sequence:

$$\begin{aligned} S_1 &= H \cup \{LS(SMA, UCC) \prec LP(UCC, SMA), \\ &\quad\ \neg LP(UCC, SMA) \prec LS(SMA, UCC)\} \\ S_2 &= S_1 \cup \{SMA \prec UCC, \neg UCC \prec SMA\} \\ S_3 &= S_2 \cup \{\neg perfected\} \\ S_4 &= S_3 \end{aligned}$$

This example nicely illustrates how in our approach conflict resolution strategies can be specified declaratively, by simply asserting relevant preferences among the involved conflicting rules.

5 Conclusions

We presented in this paper an extension of logic programs with two types of negation where preference information among rules can be expressed in the logical language. This extension is very useful for practical applications, as was demonstrated using an example from legal reasoning. In the full paper we show that the well-founded conclusions of prioritized logic programs can be computed in polynomial time. The main advantage of our approach is that also preference information is context-dependent and can be reasoned upon and derived dynamically.

[6] In realistic settings one would again use a schema here. In order to keep the example simple we use the relevant instance of the schema directly.

Acknowledgements

I would like to thank Franz Baader, Jürgen Dix, Tom Gordon, Henry Prakken and Cees Witteveen for interesting comments which helped to improve the quality of this paper.

References

[1] Baral, C., Gelfond, M., Logic Programming and Knowledge Representation, Journal of Logic Programming, 19,20:73-148, 1994

[2] Baral, C., Subrahmanian, V.S., Duality between Alternative Semantics of Logic Programs and Nonmonotonic Formalisms, Intl. Workshop on Logic Programming and Nonmonotonic Reasoning, 1991

[3] Brewka, Gerhard, Adding Priorities and Specificity to Default Logic, Proc. JELIA-94, York, 1994

[4] Brewka, Gerhard, Reasoning About Priorities in Default Logic, Proc. AAAI-94, Seattle, 1994

[5] van Gelder, A., Ross, K., Schlipf, J., The Well-Founded Semantics for General Logic Programs, Journal of ACM, p.221-230, 1990

[6] Gelfond, M., Lifschitz, V., Logic Programs with Classical Negation, Proc. 7th Intl. Conference on Logic Programming, 1990

[7] Gordon, Thomas F., The Pleadings Game: An Artificial Intelligence Model of Procedural Justice, Dissertation, TU Darmstadt, 1993

[8] Konolige, K., Hierarchic Autoepistemic Theories for Nonmonotonic Reasoning, Proc. AAAI, 1988

[9] Lifschitz, V., Computing Circumscription, Proc. 9th Int. Joint Conference on Artificial Intelligence, 1985

[10] Lifschitz, V., Foundations of Declarative Logic Programming, in: G. Brewka (ed.), Principles of Knowledge Representation, Studies in Logic, Language and Information, CSLI publishers, to appear 95

International Conference on Principles of Knowledge Representation and Reasoning, 1991

[11] Pereira, L.M., Alferes, J.J., Well Founded Semantics for Logic Programs with Explicit Negation, Proc. 10th European Conference on Artificial Intelligence, Vienna, 1992

[12] Poole, David, On the Comparison of Theories: Prefering the Most Specific Explanation, Proc. IJCAI-85, Los Angeles, 1985

[13] Prakken, Henry, Logical Tools for Modelling Legal Argument, dissertation, VU Amsterdam, 1993

[14] Przymusinski, T., The Well-Founded Semantics Coincides with the Three-Valued Stable Semantics, Fundamenta Informaticae, 1989

[15] Przymusinski, T., Stable Semantics for Disjunctive Programs, New Generation Computing, 9:401-424, 1991

[16] Reiter, Raymond, A Logic for Default Reasoning, Artificial Intelligence 13 (1980) 81-132.

[17] Tarski, Alfred, A lattice-theoretical fixpoint theorem and its applications, Pacific Journal of Mathematics, 5:285-309, 1955

[18] Touretzky, David S., The Mathematics of Inheritance, Pitman Research Notes in Artificial Intelligence, London, 1986

[19] Touretzky, David S., Thomason, Richmond H., Horty, John F., A Skeptic's Menagerie: Conflictors, Preemptors, Reinstaters, and Zombies in Nonmonotonic Inheritance, Proc. 12th IJCAI, Sydney, 1991

[20] C. Witteveen, Partial Semantics for Truth Maintenance, in: J. van Eijck (ed), Logics in AI, Springer, Heidelberg, LNAI 478, 1991, pp. 544-561