Reasoning with Prioritized Defaults

Michael Gelfond and **Tran Cao Son**
Computer Science Department
University of Texas at El Paso
El Paso, Texas 79968
mgelfond,tson@cs.utep.edu

Abstract

The purpose of this paper is to investigate the methodology of reasoning with prioritized defaults in the language of logic programs under the answer set semantics. We present a domain independent system of axioms, written as an extended logic program, which defines reasoning with prioritized defaults. These axioms are used in conjunction with a description of a particular domain encoded in a simple language allowing representation of defaults and their priorities. Such domain descriptions are of course domain dependent and should be specified by the users. We give sufficient conditions for consistency of domain descriptions and illustrate the use of our system by formalizing various examples from the literature. Unlike many other approaches to formalizing reasoning with priorities ours does not require development of a new semantics of the language. Instead, the meaning of statements in the domain description is given by the system of (domain independent) axioms. We believe that in many cases this leads to simpler and more intuitive formalization of reasoning examples. We also present some discussion of differences between various formalizations.

1 Introduction

The purpose of this paper is to investigate the methodology of reasoning with prioritized defaults in the language of logic programs under the answer set semantics. Information about relative strengths of defaults can be commonly found in natural language descriptions of various domains. For instance, in legal reasoning it is often used to state preference of some laws over others, e.g., federal laws in the U.S. can, in some cases, override the laws of a particular state. Preferences are also used in reasoning with expert's knowledge where they are assigned in accordance with the degree of our confidence in different experts. Sometimes preferences in the natural language description of the domain are given implicitly, e.g., a conflict between two contradictory defaults can be resolved by selecting the one which is based on more specific information. All these examples suggest that it may be useful to consider knowledge representation languages capable of expressing preferences between defaults. There are several possible ways to address the design of such languages. The most popular one seems to consist in developing "prioritized" versions of the existing non-monotonic formalisms [1, 2, 3, 4, 13, 12]. These extensions have few restrictions on the way priorities can be assigned to defaults and, as a result, are forced to employ comparatively complex strategies for resolving conflicting arguments according to these priorities. These strategies differ for different approaches and, as it often happens in development of knowledge representation languages, we are left with the problem of selecting one or several "best" candidates.

This paper is based on the hypothesis that one of the reasons for these difficulties may be the collective lack of experience in representing knowledge about preferences. It is often unclear for instance, if unintuitive answers to queries given by various formalisms can be blamed on the formalism itself or on the inadequate representation of the original problem. Moreover, it is often unclear what is the common-sense natural language description of the original problem of which the corresponding formal theory claims to be a representation.

To remedy the problem we propose to limit our attention to comparatively simple ways of assigning preferences to defaults and to developing a simple language $\mathcal{L}$ capable of expressing and reasoning with such preferences. We use logic programming under answer set semantics [8] as an underlying logic of $\mathcal{L}$. Theories of $\mathcal{L}$ will consist of a user defined domain description and the domain independent part consisting of axioms defining the use of prioritized defaults. The use of the language will be illustrated by various examples from the literature. All the examples were run using the SLG inference engine [5, 6].

2 The Language of Prioritized Defaults

We start with describing the class $\mathcal{L}(\sigma)$ of languages used for representing various domains of discourse. $\mathcal{L}(\sigma)$ is parameterized by a multi-sorted signature σ containing names for objects, functions and relations of the user's domain. We assume that σ contains a collection $d_1, d_2, \dots$ of function and object constants used for naming defaults. Terms of σ which have the form $d(t_1, \dots, t_n)$ where n is the arity of d and t's are terms of σ will be called *defaults*. Similar collection $r_1, r_2, \dots$ will be used to name rules. Domain knowledge in $\mathcal{L}(\sigma)$ will be described by a collection of literals of σ (called σ-literals) together with descriptions of rules, defaults, exceptions to defaults, and preferences between defaults. The syntax of such descriptions is given by the following definitions:

Definition 2.1

- σ-literals are literals of $\mathcal{L}(\sigma)$;
- if d, d_1, d_2 are defaults names, $l_0, \dots, l_n$ are literals of $\mathcal{L}(\sigma)$ and [ ] is the list operator of Prolog then

$$rule(r, l_0, [l_1, \dots, l_m]) \tag{1}$$

$$default(d, l_0, [l_1, \dots, l_m], [l_{m+1}, \dots, l_n]) \tag{2}$$

$$exception(d, [l_1, \ldots, l_m], [l_{m+1}, \ldots, l_n]) \tag{3}$$

$$prefer(d_1, d_2) \tag{4}$$

are literals of $\mathcal{L}(\sigma)$.

Sets of ground literals of $\mathcal{L}(\sigma)$ will be called *domain descriptions* (with underlying signature σ).

Domain descriptions from our examples will contain non-ground literals, which are viewed as shorthands for their ground instantiations. The relations $default$, $rule$, $exception$ and $prefer$ will be called *domain independent.* Intuitively, the statement (1) is a description of the rule r which says that if literals $l_1, \ldots, l_m$ are true in domain description $\mathcal{D}$ then so is literal l_0. l_0 is called the head of r and $l_1, \ldots, l_m$ is referred to as $r's$ body. The statement (2) is a description of the default d which says that *normally*, if $l_1, \ldots, l_m$ are true in $\mathcal{D}$ and $l_{m+1}, \ldots, l_n$ cannot be shown to be true in $\mathcal{D}$ then l_0 is true in $\mathcal{D}$. $l_1, \ldots, l_m$ are called positive preconditions of d while $l_{m+1}, \ldots, l_n$ are called negative preconditions. Both sets of preconditions will be sometimes referred to as the body of statement (2). A similar notion of body can be defined for statement (3) which prohibits application of the default d if the body of 3 may hold in $\mathcal{D}$. Finally, the statement (4) stops the application of default d_2 if the default d_1 may be applicable in $\mathcal{D}$.

This informal explanation of the meaning of domain independent relations of $\mathcal{L}(\sigma)$ will be replaced by the precise definition in the next section. But first we will attempt to clarify this meaning with the following example.

Example 2.1 Let us assume that we are given a complete list of students enrolled in various university departments, and that we know that in general, students can not write computer programs, that computer science (CS) students do it regularly and that students in computer information systems (CIS) may or may not know how to program; let us represent this information by the domain description $\mathcal{D}_0$.

The underlying signature σ of $\mathcal{D}_0$ contains student names, *mary, mike, sam, ...*, department names *cs, cis, art, ...*, appropriately typed predicate symbols $is\_in(S, D)$ and $can\_progr(S)$ read as "Student S is in department D" and "Student S can program", and two default names $d_1(S)$ and $d_2(S)$.

The defaults from our informal description can be represented by statements

$$default(d1(S), \neg can\_progr(S), [student(S)], [\,]). \tag{5}$$

$$default(d2(S), can\_progr(S), [student(S), is\_in(S, cs)], [\,]). \tag{6}$$

The last statement of the informal description suggests that the first default shall not be applicable to the students from CIS, which can be recorded as follows:

$$exception(d1(S), [is\_in(S, cis)], [\,]). \tag{7}$$

Finally, the list of students mentioned in the informal description will be represented by the collection F of facts:

$$\begin{array}{lll} student(mary). & dept(cs). & is\_in(mary, cs). \\ student(mike). & dept(art). & is\_in(mike, art). \\ student(sam). & dept(cis). & is\_in(sam, cis). \\ \ldots & \ldots & \ldots \end{array}$$

and the closed world assumption [15] for is_in, written as the default

$$default(d3(S, D), \neg is\_in(S, D), [\,], [\,]). \tag{8}$$

Relations $student$ and $dept$ are, of course, not necessary. They are playing the role of types and will later allow us to avoid floundering when applying the SLG inference engine to this example.

Informally, the domain description above should allow us to conclude that Mike does not know how to program, while we should remain undecided about programming skills of Mary and Sam. This is the case only as long as we do not assume that the second default overrides the first one, due to the specificity principle. We can use the relation $prefer$ from our language to record this preference by stating

$$prefer(d2(X), d1(X)). \tag{9}$$

Now, we can conclude that Mary can write programs, while whether Sam can write programs remains undecided.

It may be instructive to see what would happen with our conclusions if the closed world assumption for is_in were replaced by the following statement: "Normally, students belong to at most one department". The corresponding domain description $\mathcal{D}_1$ is obtained from $\mathcal{D}_0$ by replacing 8 by

$$default(d3(S, D_1), \neg is\_in(S, D_1), [is\_in(S, D_2), D_1 \neq D_2], [\,]). \tag{8$'$}$$

(Here by $=$ we mean identity of terms).

First, let us notice that our knowledge of programming abilities of Mary, Mike and Sam did not change. Since, say, Mike is in Art, by default $d_3(mike, cis)$ we conclude that he is not in CIS, hence the body of statement (7) can not be true in $\mathcal{D}_1$. Similar argument shows that the body of default $d_2(mike)$ can not be true in $\mathcal{D}_1$ either and hence default $d_1(mike)$ can be applied to conclude that Mike cannot program. The situation changes if we have a student Bob whose affiliation with the university departments is not known, i.e., the only statement about Bob in $\mathcal{D}_1$ is

$$student(bob).$$

Notice that now we cannot conclude that Bob is not in CIS and therefore, in accordance with the intuitive meaning of statement (7), the application of the default $d1(bob)$ is stopped and we remain undecided about his programming abilities. In the same situation, $\mathcal{D}_0$ will of course conclude that Bob does not know how to program since the prerequisites of (6) and (7) will be invalidated by the CWA.

The next example demonstrates the use of negative preconditions of defaults.

Example 2.2 *[14]* Consider the following two defaults from the domain description $\mathcal{D}_2$ taken from the legal reasoning domain:

1. A person who cannot be shown to be a minor has the capacity to perform legal acts.
2. Any person can exercise the right to vote if he/she is not a minor.

The first default can be represented by the default description

$default(d_1(x), has\_legal\_capacity(x), [\,], [minor(x)])$

which requires a negative precondition. The second default has the form

$default(d_2(x), has\_right\_to\_vote(x), [\neg minor(x)], [\,])$

These defaults, used in conjunction with statement

$\neg minor(jim)$,

entail that Jim has legal capacity and can vote. If the system is asked the same question about Mary whose legal age is not known it will conclude that Mary has legal capacity but will remain in the dark about Mary's right to vote.

The last example in this section is meant to illustrate the behavior of conflicting defaults and the use of a preference relation between them.

Example 2.3 Consider the domain description $\mathcal{D}_3$ consisting of two defaults

$default(d_1, p, [], [])$,

$default(d_2, q, [r], [])$,

the rule

$rule(r_1, \neg p, [q])$

and the fact r. The intended meaning of $\mathcal{D}_3$ should sanction two alternative sets of conclusions: one, containing p and another containing q and $\neg p$. If we expand $\mathcal{D}_3$ by

$prefer(d_2, d_1)$

the application of d_1 should be blocked and the domain description $\mathcal{D}_3$ should entail q and $\neg p$. Notice that according to our interpretation of the preference relation default d_1 would be blocked even if the fact r were removed from the description. In this case the new description would entail neither p, nor q, nor $\neg p$.

More examples of the use of the language $\mathcal{L}$ for describing various domains will be found in the following sections. But, let us first give a precise definition of entailment from domain descriptions which captures the intended meaning of statements of $\mathcal{L}$.

2.1 Axioms of $\mathcal{P}$

In this section we present a collection $\mathcal{P}_\sigma$ of axioms defining the meaning of the domain independent relations of $\mathcal{L}(\sigma)$. The axioms are stated in the language of logic programs under the answer set semantics. They are intended to be used in conjunction with domain descriptions of $\mathcal{L}(\sigma)$ and to define the collection of statements which (strictly and/or defeasibly) follow from the given domain description $\mathcal{D}$. More precisely, we consider two basic relations $holds(l)$ and $holds\_by\_default(l)$ defined on literals of $\mathcal{L}(\sigma)$ which stand for "strictly holds" and "defeasibly holds", respectively. The query language associated to domain descriptions of $\mathcal{L}(\sigma)$ will consist of ground atoms of the form $holds\_by\_default(l)$, $holds(l)$, and their negations.

Definition 2.2 We say that a domain description $\mathcal{D}$ entails query q ($\mathcal{D} \models q$) if q belongs to every answer set of the program $\mathcal{P}_\sigma(\mathcal{D}) = \mathcal{P}_\sigma \cup \{holds(l) \mid l \in \mathcal{D}\}$.

Program $\mathcal{P}$[1] consists of the following rules:

Non-defeasible Inference:

$$\begin{aligned} holds(L) \;\leftarrow\;& holds(rule(R, L, Body)), \\ & hold(Body). \end{aligned} \tag{10}$$

$$hold([\,]). \tag{11}$$

$$\begin{aligned} hold([H|T]) \;\leftarrow\;& holds(H), \\ & hold(T). \end{aligned} \tag{12}$$

The first axiom defines the relation $holds$ which is satisfied by a $\mathcal{L}(\sigma)$ literal l iff l is non-defeasibly true in the domain description $\mathcal{D}$. The next two axioms define the same relation on the lists of literals in $\mathcal{L}(\sigma)$, i.e., $hold([l_1, \dots, l_n])$ iff all l's from the list are true in $\mathcal{D}$.

Defeasible Inference:

$$holds\_by\_default(L) \;\leftarrow\; holds(L). \tag{13}$$

$$\begin{aligned} holds\_by\_default(L) \;\leftarrow\;& holds(rule(R, L, Body)), \\ & hold\_by\_default(Body). \end{aligned} \tag{14}$$

$$holds\_by\_default(L) \;\leftarrow\; holds(default(D, L, Positive, Negative)), \tag{15}$$

[1] In what follows we assume that σ is fixed and omit reference to it whenever possible.

$$\begin{aligned} & hold\_by\_default(Positive), \\ & fail\_by\_default(Negative), \\ & not\ defeated(D), \\ & not\ \neg holds\_by\_default(L). \end{aligned}$$

$$hold\_by\_default([\,]). \tag{16}$$

$$hold\_by\_default([H|T]) \leftarrow holds\_by\_default(H), \; hold\_by\_default(T). \tag{17}$$

$$fail\_by\_default([\,]). \tag{18}$$

$$fail\_by\_default([H|T]) \leftarrow not\ holds\_by\_default(H), \; fail\_by\_default(T). \tag{19}$$

The first axiom in this group ensures that strictly true statements are also true by default. The next one allows application of rules for defeasible inference. The third axiom states that defaults with proven premises imply their conclusions unless they are defeated by other rules and defaults of the domain description. The condition $not\ \neg holds\_by\_default(L)$ is used when the domain contains two undefeated defaults D_1 and D_2 with conflicting conclusions (like in Example 2.3). In this case $\mathcal{P}(\mathcal{D})$ will have multiple answer sets, one containing the conclusion of D_1 and the other containing the conclusion of D_2. The alternative solution here is to stop applications of both defaults, but we believe that in some circumstances (like those described by the extended *"Nixon Diamond"*) our solution is preferable.

The last four rules from this group define relation $hold\_by\_default(List)$ which holds if all literals from the list hold by default and relation $fail\_by\_default(List)$ which holds if no literal from the list holds by default.

Defeating defaults:

$$defeated(D) \leftarrow holds(exception(D, Positive, Negative)), \; may\_hold(Positive), \; fail\_by\_default(Negative). \tag{20}$$

$$defeated(D) \leftarrow holds(default(D, L, Positive, Negative)), \; \neg holds(L). \tag{21}$$

$$defeated(D) \leftarrow holds(default(D1, L, Positive, Negative)), \tag{22}$$

$$\begin{aligned} & holds\_by\_default(prefer(D1, D)), \\ & may\_hold(Positive), \\ & fail\_by\_default(Negative), \\ & not\ defeated(D_1). \end{aligned}$$

$$may\_hold([\,]). \tag{23}$$

$$may\_hold([H|T]) \leftarrow not\ \neg hold\_by\_default(H), \; may\_hold(T). \tag{24}$$

These axioms describe several possible ways to defeat a default d. The first axiom stops the application of default d if d has an exception whose precondition might be satisfied by the domain description. The next axiom describes a stronger type of defeat when the conclusion of the default is proven to be false by non-defeasible means. The axiom (22) allows defeating of d by undefeated defaults of higher priority. Lastly, *may_hold* explains itself.

Auxiliary:

$$\neg holds(L) \leftarrow holds(\neg L). \tag{25}$$

$$\neg holds\_by\_default(L) \leftarrow holds\_by\_default(\neg L). \tag{26}$$

These auxiliary axioms are self-explanatory.

We believe that $\mathcal{P}(\mathcal{D})$ captures a substantial part of our intuition about reasoning with prioritized defaults and therefore deserves some study. Its properties obviously depend on the properties of $\mathcal{D}$ so we should probably start with some classification of domain descriptions. The following simple definition gives a first classification:

Definition 2.3 We will say that a domain description $\mathcal{D}$ is *consistent* if $\mathcal{P}(\mathcal{D})$ is consistent, i.e., has a consistent answer set.

Obviously, not all domain descriptions are consistent. $\mathcal{D} = \{p, \neg p\}$, for instance, is not. Sometimes however inconsistency is more subtle. In the next example it is caused not by direct contradiction but by the absence of answer sets of $\mathcal{P}(\mathcal{D})$, i.e., by the presence of meaningless defaults.

Example 2.4 Consider the domain description $\mathcal{D}_4 = \{default(d, p, [], [p])\}$. It is easy to see that $\mathcal{P}(\mathcal{D}_4)$ contains the following rule:

$$holds\_by\_default(p) \leftarrow not\ defeated(d), not\ holds\_by\_default(p), not\ \neg holds\_by\_default(p).$$

Because of this rule, $\mathcal{P}(\mathcal{D}_4)$ does not have an answer set. Thus, $\mathcal{D}_4$ is inconsistent.

In general, $\mathcal{P}(\mathcal{D})$ has multiple answer sets. For instance, it is easy to check that the program $\mathcal{P}(\mathcal{D}_3)$ from 2.3 has two answer sets containing $\{holds\_by\_default(p)\}$ and $\{holds\_by\_default(q), \neg holds\_by\_default(p)\}$ respectively. It is natural to look for sufficient condition guaranteeing consistency, categoricity (uniqueness of answer set) and other useful properties of $\mathcal{P}(\mathcal{D})$. In this paper we concentrate on consistency of domain descriptions. The Proposition 2.1 below serves as an example of the issues involved. We start with the following definitions:

Definition 2.4 A domain description $\mathcal{D}$ is called *simple* if

1. $\mathcal{D}$ does not contain rules;
2. each statement of $\mathcal{D}$ contains at most one occurrence of domain independent predicate symbols;
3. the program consisting of $\{holds(l) \mid l \in \mathcal{D}\}$ and rule (25) is consistent;
4. $\mathcal{D}$ does not contain a sequence of statements of the form $prefer(d_0, d_1), prefer(d_1, d_2) \dots prefer(d_n, d_0)$.

Definition 2.5 A domain description $\mathcal{D}$ is said to be *acyclic* if there exists a function α which assigns to each literal l of $\mathcal{L}(\sigma)$ an ordinal $\alpha(l)$ such that

1. for every literal l of $\mathcal{L}(\sigma)$, $\alpha(l) = \alpha(\neg l)$;
2. for every default description $default(d, l, pos, neg)$ in $\mathcal{D}$,
   (a) $\alpha(d) = \alpha(l)$ and
   (b) $\alpha(l) > max\{\alpha(l') \mid l' \in pos \cup neg\}$;
3. for every statement of the form $rule(r, l, body)$ in $\mathcal{D}$,
   (a) $\alpha(r) = \alpha(l)$ and
   (b) $\alpha(l) > max\{\alpha(l') \mid l' \in body\}$;
4. for every statement of the form $exception(d, pos, neg)$ in $\mathcal{D}$, $\alpha(head(d)) > max\{\alpha(l') \mid l' \in pos \cup neg\}$;
5. for every statement of the form $prefer(d', d))$ in $\mathcal{D}$, $\alpha(d') \leq \alpha(d)$.

It is easy to see that the domain description $\mathcal{D}_0$ from Example 2.1 is acyclic while $\mathcal{D}_1$ (from the same example) is not acyclic.

Proposition 2.1 For every simple and acyclic domain description $\mathcal{D}$, the program $\mathcal{P}(\mathcal{D})$ has a consistent answer set.

3 Using the Axioms

In this section we illustrate the use of our approach by formalizing several examples of reasoning with priorities. In what follows we will often refer to running our programs using SLG inference engine. Strictly speaking, the SLG does not allow "$\neg$" in its syntax. The problem is avoided by treating $\neg$ as a new function symbol, writing rules defining contrary literals and considering only those stable models of $\mathcal{P}(\mathcal{D})$ which do not contain such both literals a and *neg(a)*, as demonstrated in the following example.

Example 3.1 Assume that we have two defaults:

(1) "normally a holds" and

(2) "normally $\neg a$ holds."

We also know that b holds and there is a rule which says that if $\neg a$ holds then so $\neg b$.

The above information can be then represented as the domain description $\mathcal{D}_5$:

$$
\begin{array}{l}
b. \\
default(d1, a, [\,], [\,]). \\
default(d2, neg(a), [\,], [\,]). \\
rule(r1, neg(b), [neg(a)]).
\end{array}
$$

Given this domain description SLG returns two stable models S_1 and S_2 containing $\{holds\_by\_default(a), holds\_by\_default(b)\}$ and $\{\neg holds\_by\_default(a), \neg holds\_by\_default(b), holds\_by\_default(b)\}$ respectively. Since S_2 contains contrary literals the only answer set of $\mathcal{P}(\mathcal{D}_5)$ is S_1, i.e., $holds\_by\_default(a)$ and $holds\_by\_default(b)$ are entailed by $\mathcal{P}(\mathcal{D}_5)$.

Now let us go back to Example 2.1.

Example 3.2 *(Example 2.1 revisited)* It is easy to see that domain description $\mathcal{D}_6$ consisting of statements (5)-(8) and the facts from Example 2.1 has two answer sets, containing $\{\neg holds\_by\_default(can\_progr(mary)), \neg holds\_by\_default(can\_progr(mike))\}$ and $\{holds\_by\_default(can\_progr(mary)), \neg holds\_by\_default(can\_progr(mike))\}$, respectively. Notice, that none of the answer sets contains information about Sam's programming ability. Hence, we can conclude that Mike does not know how to program but we have to stay undecided on the same question about Mary and Sam.

If we expand the domain by adding to it statement (9) about the preference of $d2$ over $d1$ then the first answer set will disappear which of course corresponds exactly to our intention. It may be instructive to expand our domain by the following information: "Bad students never know how to program. Bob is a bad computer science student". This can be represented by facts

$$
\begin{array}{l}
student(bob). \\
bad(bob). \\
is\_in(bob, cs).
\end{array}
$$

and the rule

$rule(r_2(S), \neg can\_progr(S), [student(S), bad(S)])$

The new domain description will correctly entail that Bob does not know how to program. Notice, that if the above rule were changed to the default

$default(d_3(S), \neg can\_progr(S), [student(S), bad(S)], [\,])$

we would again get two answer sets with contradictory conclusions about Bob, and that again the conflict could be resolved by adding, say,

$prefer(d_3(S), d_2(S))$.

It is equally easy to check that variations of this example also yield the intended results.

The previous example had an introductory character and could have been nicely formalized without using the preference relation. The next example (from [2], which attributes it to [9]) is more sophisticated: Not only does it require the ability to apply preferences to resolve conflicts between defaults, but also the ability of using defaults to *reason about such preferences*. Brewka in [2] argues that the ability to reason about preferences between defaults in the same language in which defaults are stated is important for various applications. In legal reasoning similar arguments were made by Gordon, Prakken, and Sartor [9, 13]. On the other hand, many formalisms developed for reasoning with prioritized defaults treat preferences as something statically given and specified separately from the corresponding default theory.

Example 3.3 *(Legal Reasoning [2])* Assume that 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 a security interest in goods may be perfected by taking possession of the collateral. However, there is a federal law called 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 preference to newer law. 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.

Let us build the domain description $\mathcal{D}_7$ which represents the above information. We will follow the formalization from [2] which uses symbols $possession$ for "ship is a possession of the lady from the above story", $perfected$ for "the ownership of the ship is perfected", and $filed$ for "financial statement about possession of the ship is filed". The domain also contains legal laws represented by default names and function symbols *state(D), federal(D), more_recent(D_1,D_2)* and *conflict(D_1,D_2)*. Here, $conflict(D_1, D_2)$ means that default D_1 and default D_2 are in conflict and therefore cannot be applied simultaneously.

The UCC and SMA defaults of $\mathcal{D}_7$ can be represented by

$$default(d_1, perfected, [possession], [\,]).$$
$$default(d_2, neg(perfected), [neg(filed)], [\,]).$$

The two legal principles for resolving conflicts are represented by the next two defaults:

$$default(d_3(D_1, D_2), prefer(D_1, D_2), [more\_recent(D_1, D_2), conflict(D_1, D_2)], [\,]).$$
$$default(d_4(D_1, D_2), prefer(D_1, D_2), [federal(D_1), state(D_2), conflict(D_1, D_2)], [\,]).$$

Notice, that these laws assign preferences only to the conflicting laws. We adopt a simple notion of conflicting laws given by the following statement:

$$rule(r_1(D_1, D_2), conflict(D_1, D_2), [default(D_1, L_0, B_1), default(D_2, \overline{L_0}, B_2)])$$

where of course $\overline{a} = neg(a)$ and $\overline{neg(a)} = a$. The next defaults will express the closed world assumptions for relations $more\_recent, federal$ and $state$. Presumably, a reasoning legal agent must have complete knowledge about the laws.

$$default(d_5(D_1, D_2), \neg more\_recent(D_1, D_2), [\,], [\,]).$$
$$default(d_6(D), \neg federal(D), [\,], [\,]).$$
$$default(d_7(D), \neg state(D), [\,], [\,]).$$

To complete our formalization we need the following facts:

$neg(filed).$
$possession.$
$more\_recent(d_1, d_2).$
$federal(d_2).$
$state(d_1).$

It is not difficult to check (using SLG if necessary) that the program $\mathcal{P}(\mathcal{D}_7)$ has two answer sets where

(i) $holds\_by\_default(perfected)$

belongs to one answer set and

(ii) $\neg holds\_by\_default(perfected)$

belongs to the other. This is because we have two defaults d_1 and d_2: the former supports the first conclusion, the latter - the second one, and preference between them cannot be resolved using defaults d_3 and d_4. Thus, neither (i) nor (ii) is entailed by $\mathcal{P}(\mathcal{D}_7)$. This is also Brewka's result in [2].

However, if we know that d_4 has a preference over d_3, the situation changes; To see that, let us expand our domain description by

$$prefer(d4(D_1, D_2), d3(D_2, D_1)).,$$

and denote the new domain description by $\mathcal{D}'_7$; as a result, program $\mathcal{P}(\mathcal{D}'_7)$ has then only one answer set, which contains (ii). This is again the desired behavior, according to [2]. It may be worth noticing that the closed world assumptions d_5, d_6 and d_7 have no role in the above arguments and could be removed from the domain description. They are important, however, for its general correctness. The example can be substantially expanded by introducing more realistic representation of the story and by using more complex strategies of assigning preferences to conflicting defaults. We argue that the corresponding domain descriptions will remain natural and correct.

We use the next example from Brewka [4] to illustrate differences between our theory and several other formalisms dealing with prioritized defaults.

Example 3.4 *(Gray Area)* Brewka considers the following defaults:

1. "Penguins normally do not fly;"
2. "Birds normally fly;" and
3. "Birds that can swim are normally penguins;"

under the assumption that default (1) is preferred over (2), and (2) preferred over (3). (Notice, that Brewka assumes transitivity of the preference relation).

These defaults are represented in his formalism by the program

$bird.$
$swims.$
$(d1)\ \neg flies \leftarrow not\ flies, penguin.$
$(d2)\ flies \leftarrow not\ \neg flies, bird.$
$(d3)\ penguin \leftarrow not\ \neg penguin, bird, swims.$

According to Brewka, the prioritized default theories from [1, 2, 12] are applicable to this case and produce single extension $E_1 = \{swims,\ bird,\ flies,\ penguin\}$ which is certainly contrary to intuition. His own theory in [4] produces the extension $E_2 = \{swims,\ bird,\ \neg\ flies,\ penguin\}$ which is, in our opinion, a somewhat better, but not completely satisfactory behavior, since we do not see why application of the default $(d3)$ is not stopped by more powerful defaults $(d1)$ and $(d2)$. We can suggest the following line of reasoning we believe should be accepted as valid in this case. We are initially confronted with "ready to fire" defaults $(d2)$ and $(d3)$. Since $(d2)$ has a higher priority it wins and we conclude $flies$. $(d3)$ is still blocked by $(d2)$, $(d1)$ is not applicable and hence we stop.

Alternatively, we can apply defaults $(d1)$ and $(d3)$. Since $(d2)$ is then defeated by $(d1)$ it will not block $(d3)$. The corresponding conclusions shall then contain $penguin$ and $\neg flies$. But this will work only if $(d1)$ is not preferred to $(d3)$. If it is then $(d1)$, whose body is satisfied, should block the application of $(d3)$ and hence we have no way of obtaining $penguin$.

These lines of reasoning are captured in our approach. The information above is naturally encoded in the domain description $\mathcal{D}_8$ by the following statements

$default(d_1, \neg flies, [penguin], [\,]).$
$default(d_2, flies, [bird], [\,]).$
$default(d_3, penguin, [bird, swims], [\,]).$

$prefer(d_1, d_2).$
$prefer(d_2, d_3).$
$prefer(d_1, d_3).$
$bird.$
$swims.$

The program $\mathcal{P}(\mathcal{D}_8)$ has only one answer set which contains

$$\{holds\_by\_default(bird), holds\_by\_default(swims), holds\_by\_default(flies)\}.$$

This is because d_3 is blocked by d_2 and thus, d_1 cannot be applied due to the fact that $holds\_by\_default(penguin)$ does not hold.

However, if we remove the statement $prefer(d_1, d_3)$ from $\mathcal{D}_8$ then $\mathcal{P}(\mathcal{D}'_8)$ has two answer sets: the first one is the same as above and the second one contains statements $holds\_by\_default(bird)$, $holds\_by\_default(swims)$, $\neg holds\_by\_default(flies)$ and $holds\_by\_default(penguin)$.

We realize of course that this example belongs to the *gray area* and can be viewed differently. The main lesson from this observation is that in the process of expressing ourselves (while programming or otherwise) we should try to avoid making unclear statements. Of course, we hope that further work on semantics will help to clarify some statements which so far remain unclear. We also hope that the reader is not left with the impression that we claim success in following our own advice.

This paper is a report on the work which is still in progress. In the full paper we plan to further investigate properties of our system, expand it in several directions and to do more complete comparison of our approach with existing approaches to the problem of reasoning with priorities.

Acknowledgment

We are grateful to Gerhard Brewka for an illuminating discussion on reasoning with prioritized defaults. We also would like to thank Alfredo Gabaldon for useful discussions and help with running our examples on SLG.

References

[1] Baader, F. and Hollunder, B.: Priorities on Defaults with Prerequisite and their Application in Treating Specificity in Terminological Default Logic, Journal of Automated Reasoning, 15:41–68, 1995.

[2] Brewka, G.: Reasoning about Priorities in Default Logic, Proc. AAAI-94, Seattle, 1994

[3] Brewka, G.: Adding Priorities and Specificity to Default Logic, Proc. JELIA 94, Springer LNAI 838, 247–260, 1994

[4] Brewka, G.: Preferred Answer Sets, unpublished manuscript, 1997.

[5] Chen, W. and Warren, D.S.: Query Evaluation under the Well-Founded Semantics, The Twelfth ACM Symposium on Principles of Database System, 1993.

[6] Chen, W.: Extending Prolog with Nonmonotonic Reasoning, Journal of LP, 169–183, 1996.

[7] Dung, P.M.: On the Relations Between Stable and Well-Founded Semantics of Logic Programming, Theoretical Computer Science 105:7-25 (1992).

[8] Gelfond, M. and Lifschitz, V.: Logic Programs with Classical Negation, Proceeding of the 7th ICLP, MIT Press, 579 –597, 1990. (Also in "New Generation of Computing", 1991.)

[9] Gordon, T.: The Pleadings Game: An Artificial Intelligence Model of Procedural Justice. Ph.D. Dissertation, TU Darmstadt.

[10] Fages, F.: Consistency of Clark's Completion and Existency of Stable Models, Technical Report 90-15, Ecole Normale Superieure, 1990.

[11] Lifschitz, V. and Turner, H.: Splitting a Logic Program, Proc. of ICLP, MIT Press, 1994.

[12] Marek, W. and Truszczynski, M.: Nonmonotonic Logic: Context-Dependent Reasoning, Springer, 1993.

[13] Prakken, H. and Sartor, G.: On the relation between legal language and legal argument: assumptions, applicability and dynamic priorities. Proc. of the Fifth International Conference on AI and Law, AAAI, College Park, MD USA, 1–10, 1995.

[14] Prakken, H. and Sartor, G.: Argument-based extended logic programming with defeasible priorities. Journal of applied non-classical logics, 1,2 (7), 25–77, 1997.

[15] Reiter, R.: On closed world data bases. In H. Gallaire and J. Minker, editors, Logic and data bases, 55–76, 1978.

Proof of Proposition 2.1

Here we give a proof of Proposition 2.1. We assume that the reader is familiar with notation and the results from [7], [10], [8], [11].

Proposition 2.1 For every simple and acyclic domain description $\mathcal{D}$, the program $\mathcal{P}(\mathcal{D})$ has a consistent answer set.

Proof: Let

$$\begin{aligned} U = \; & \{holds(L) \mid L \text{ is literal of } \mathcal{L}(\sigma)\} \cup \\ & \{hold([L_1, \ldots, L_n]) \mid L_i \text{ is a literal of } \mathcal{L}(\sigma) \text{ for } i = 1, \ldots, n\} \end{aligned}$$

It is easy to see that U is a splitting set of $\mathcal{P}(\mathcal{D})$. Furthermore, $b_U(\mathcal{P}(\mathcal{D}))$, the bottom of $\mathcal{P}(\mathcal{D})$ with respect to U, consists of $\{holds(L) \mid L \in \mathcal{D}\}$ and the rules (10)-(12) and (25).

From clauses (1) and (3) of the definition 2.4 we can conclude that $b_U(\mathcal{P}(\mathcal{D}))$ has a unique consistent answer set $X_0 = X_1 \cup X_2$ where $X_1 = \{holds(L) \mid L \in \mathcal{D}\} \cup \{\neg holds(L) \mid \neg L \in \mathcal{D}\}$ and $X_2 = \{hold([L_1, \ldots, L_n]) \mid holds(L_i) \in X_1 \text{ for } i = 1, \ldots, n\}$.

Let $\pi_0 = e_U(\mathcal{P}(\mathcal{D}) \setminus b_U(\mathcal{P}(\mathcal{D})), X_0)$.

Now, notice that by clause (2) of definition 2.4, set $Pref$ of literals of the form $hold\_by\_default(prefer(d_1, d_2))$ forms a splitting set of π_0 where the bottom of π_0 with respect to $Pref$ is $Pref$ itself. Let us denote $e_{Pref}(\pi_0 \setminus Pref, Pref)$ by π.

Note that if the program π is consistent, i.e., π has a consistent answer set Y then $\langle Pref, Y\rangle$ is a consistent solution of π_0 with respect to $Pref$. It means that $X \cup Pref \cup Y$ is a consistent answer set of $\mathcal{P}(\mathcal{D})$. In the following, we will show that π is consistent and hence $\mathcal{P}(\mathcal{D})$ is consistent. $\square$

We now prove that the program π is consistent. By definition, π consists of following rules.

$$holds\_by\_default(l). \qquad \text{(for every } holds(l) \in X). \tag{13.1}$$

$$\begin{aligned} holds\_by\_default(l) \leftarrow\ & hold\_by\_default(positive), \\ & fail\_by\_default(negative), \\ & not\ defeated(d), \\ & not\ \neg holds\_by\_default(l). \end{aligned} \tag{15.1}$$

(for every $default(d, l, positive, negative) \in \mathcal{D}$)

$$hold\_by\_default([\,]). \tag{16.1}$$

$$\begin{aligned} hold\_by\_default([h|t]) \leftarrow\ & holds\_by\_default(h), \\ & hold\_by\_default(t). \end{aligned} \tag{17.1}$$

$$fail\_by\_default([\,]). \tag{18.1}$$

$$\begin{aligned} fail\_by\_default([h|t]) \leftarrow\ & not\ holds\_by\_default(h), \\ & fail\_by\_default(t). \end{aligned} \tag{19.1}$$

$$\begin{aligned} defeated(d) \leftarrow\ & may\_hold(positive), \\ & fail\_by\_default(negative). \end{aligned} \tag{20.1}$$

(for every $exception(d, positive, negative) \in \mathcal{D}$)

$$defeated(d). \qquad \text{(for every } default(d, l, positive, negative) \in \mathcal{D} \text{ where } \neg holds(l) \in X) \tag{21.1}$$

$$\begin{aligned} defeated(d) \leftarrow\ & may\_hold(positive), \\ & fail\_by\_default(negative), \\ & not\ defeated(d_1). \end{aligned} \tag{22.1}$$

(for every $default(d_1, l, positive, negative) \in \mathcal{D}$ and $prefer(d_1, d) \in \mathcal{D}$)

$$may\_hold([\,]). \tag{23.1}$$

$$may\_hold([h|t]) \leftarrow not\ \neg hold\_by\_default(h), may\_hold(t). \tag{24.1}$$

$$\neg holds\_by\_default(l) \leftarrow holds\_by\_default(\neg l). \tag{26.1}$$

Note that every literal of π is of the form $p(l)$ or $q([l_1, \dots, l_n])$ where p, q are predicate names in π such as $holds\_by\_default$, $hold\_by\_default$, etc.

Since $\mathcal{D}$ is acyclic, there exists a function α that satisfies the condition in Definition 2.5. We define a function β from $lit(\pi)$ into ordinals as follows.

- for every positive literal $p(l)$ of π, $\beta(p(l)) = \alpha(l)$ and $\beta(\neg p(l)) = \alpha(l)$;
- for every positive literal $q([l_1, \dots, l_n])$ of π, $\beta(q([l_1, \dots, l_n]) = max\{\alpha(l_1), \dots, \alpha(l_n)\}$ and $\beta(\neg q([l_1, \dots, l_n]) = max\{\alpha(l_1), \dots, \alpha(l_n)\}$.

Notation: for a set S of literals of π, let $\beta(S) = \max\{\beta(p) \mid p \in S\}$.

We can prove the following lemma.

Lemma 2.1 For every rule r in π, $\beta(head(r)) \geq \beta(body(r))$[2].

Proof: It is easy to see that the lemma is trivial for rules (13.1), (16.1)-(19.1), (21.1), and (23.1)-(26.1). We need to prove the lemma for the other rules. (i)

For rule (15.1), we know that d, l, $positive$, and $negative$ come from the default description $default(d, l, positive, negative) \in \mathcal{D}$. Hence,
$\beta(holds\_by\_default(l)) = \alpha(l)$,
$\beta(hold\_by\_default(positive)) < \alpha(l)$ (Condition 2, Definition 2.5),
$\beta(fail\_by\_default(negative)) < \alpha(l)$ (Condition 2, Definition 2.5),
$\beta(defeated(d)) = \alpha(d) = \alpha(l)$ (Condition 2, Definition 2.5), and
$\beta(\neg holds\_by\_default(l)) = \alpha(\neg l) = \alpha(l)$. (Condition 1, Definition 2.5)
It follows that $\beta(head(r)) \geq \beta(body(r))$ for rule (15.1). (ii)

Similarly, because of Condition 4 of Definition 2.5, the lemma is correct for rules (20.1). (iii)

Furthermore, because of Condition 5 of Definition 2.5, the lemma is correct for rules (22.1). (iv)

The lemma is proved by (i)-(iv). □

For an arbitrary ordinal δ, let S_δ be a set of literal of π such that

$$S_\delta = \{l \mid \beta(l) \leq \delta\}.$$

By Lemma 2.1, we have that for any rule r of π, if $head(r) \in S_\delta$ then $body(r) \subseteq S_\delta$. Therefore, $S = \langle S_\delta \rangle_{\delta < \mu}$ is a splitting sequence of π where μ is the smallest ordinal number such that for every literal $l \in lit(\pi)$, $\beta(l) < \mu$.

The next lemma shows that there exists a consistent solution of π with respect to S.

[2] $head(r)$ denotes the head of r and $body(r)$ denotes the set of literals occur in the body of r.

Lemma 2.2 There exists a consistent solution $W = \langle W_\delta \rangle_{\delta<\mu}$ of π with respect to the splitting sequence $S = \langle S_\delta \rangle_{\delta<\mu}$ of π.

Proof: To prove the lemma we will construct a solution of π with respect to S. We do that in two steps: First, we show that the program $b_{S_0}(\pi)$ has a consistent answer set W_0. Next we show that for any non limit ordinal δ, if $\delta + 1 < \mu$ then $e_{S_\delta}(b_{S_{\delta+1}}(\pi) \setminus b_{S_\delta}(\pi), \bigcup_{\eta<\delta} W_\delta)$ has a consistent answer set $W_{\delta+1}$.

1. Consider the program $b_{S_0}(\pi)$. Because of Lemma 2.1, we can conclude that there is no rules (15.1) and (20.1)-(22.1) in $b_{S_0}(\pi)$, i.e., $b_{S_0}(\pi)$ is a collection of facts together with the rules (16.1)-(19.1), (23.1)-(24.1), and (26.1). Furthermore,

   $$Y = \{holds\_by\_default(l) \mid holds(l) \in X\} \cup \{\neg holds\_by\_default(l) \mid \neg holds(l) \in X\}$$

   is consistent because X is consistent. Therefore,

   $$\begin{array}{rcl} W_0 = S_0 \cap (Y & \cup & \{hold\_by\_default([l_1, \ldots, l_n]) \mid holds\_by\_default(l_i) \in Y\} \\ & \cup & \{fail\_by\_default([l_1, \ldots, l_n]) \mid holds\_by\_default(l_i) \notin Y\} \\ & \cup & \{may\_hold([l_1, \ldots, l_n]) \mid \neg holds\_by\_default(l_i) \notin Y\}) \end{array}$$

   is consistent.

   It is easy to see that W_0 is a consistent answer set of $b_{S_0}(\pi)$. Hence, $b_{S_0}(\pi)$ has a consistent answer set. (*)

2. We now prove by transfinite induction over δ that: for every non limited ordinal δ if $\delta + 1 < \mu$ then

   (a) there exist an answer set $X_{\delta+1}$ of $e_{S_\delta}(b_{S_{\delta+1}}(\pi) \setminus b_{S_\delta}(\pi), \bigcup_{\eta\leq\delta} X_\delta)$, and

   (b) $\bigcup_{\eta\leq\delta+1} X_\eta$ is consistent.

   Since the proof for the base case and the proof for the inductive step are similar we omit here the proof for the base case.

   Inductive step: Assume that we have proved our claim for $\eta < \delta$. We now prove it for δ which is not a limited ordinal.

Let $SW = \bigcup_{\eta<\delta} W_\eta$ and π_1 be the program $e_{S_\delta}(b_{S_{\delta+1}}(\pi) \setminus b_{S_\delta}(\pi), \bigcup_{\eta<\delta} W_\eta)$. It follows from the definition of $S_{\delta+1}$ and S_δ that π_1 consists of following rules:

$$holds\_by\_default(l). \qquad \text{(for every} holds(l) \in X.) \tag{13.2}$$

$$\begin{array}{lll} holds\_by\_default(l) & \leftarrow & not\ defeated(d), \\ & & not\ \neg holds\_by\_default(l). \\ & & \text{(for every} default(d, l, positive, negative)) \in \mathcal{D}, \\ & & \text{such that } hold\_by\_default(positive) \in SW, \\ & & \text{and } fail\_by\_default(negative) \in SW.) \end{array} \tag{15.2}$$

$$\begin{array}{lll} hold\_by\_default([h|t]) & \leftarrow & holds\_by\_default(h), \\ & & hold\_by\_default(t). \end{array} \tag{17.2}$$

$$\begin{array}{lll} fail\_by\_default([h|t]) & \leftarrow & not\ holds\_by\_default(h), \\ & & fail\_by\_default(t). \end{array} \tag{19.2}$$

$$\begin{array}{ll} defeated(d). & \text{(for every } exception(d, positive, negative) \in \mathcal{D}, \\ & \text{such that } fail\_by\_default(negative) \in SW, \\ & \text{and } may\_hold(positive) \in SW.) \end{array} \tag{20.2}$$

$$\begin{array}{ll} defeated(d). & \text{(for every} default(d, l, positive, negative) \in \mathcal{D} \\ & \text{such that } \neg holds(l) \in X.) \end{array} \tag{21.2}$$

$$\begin{array}{lll} defeated(d) & \leftarrow & not\ defeated(d_1). \\ & & \text{(for every } prefer(d_1, d) \in \mathcal{D} \\ & & \text{such that } default(d_1, l, positive, negative) \in \mathcal{D}, \\ & & \qquad may\_hold(positive) \in SW, \\ & & \text{and } fail\_by\_default(negative) \in SW.) \end{array} \tag{22.2}$$

$$\begin{array}{lll} may\_hold([h|t]) & \leftarrow & not\ \neg hold\_by\_default(h), \\ & & may\_hold(t). \end{array} \tag{24.2}$$

$$\neg holds\_by\_default(l) \leftarrow holds\_by\_default(\neg l). \tag{26.2}$$

where for every rule r in π_1, $\beta(head(r)) = \delta + 1$.

Consider the general program π_1' obtained from π_1 by replacing literal of the form $\neg holds\_by\_default(l)$ with $holds\_by\_default'(l)$ in all rules of π_1 (See [8] for more detail about the transformation). Let G be the atom dependency graph of the program π_1'.

It is easy to see that there exists no cycle in G containing literal of the form $hold\_by\_default([l_1, \ldots, l_n])$, $fail\_by\_default([l_1, \ldots, l_n])$, or $may\_hold([l_1, \ldots, l_n])$. By condition (4) of definition 2.4 there exists no cycle in G which starts at the literal of the form $defeated(d)$. Hence, there are only cycles consisting of $holds\_by\_default(l)$

and $holds\_by\_default'(l)$ in G with two negative links: from $holds\_by\_default(l)$ to $holds\_by\_default'(l)$ and from $holds\_by\_default'(l)$ to $holds\_by\_default(l)$. In other words, π_1' contains only even number negative cycle. It follows from the result of Fages in [10], π_1' has a stable model A.

From a theorem in [8] we know that $A' = \{a(x) \in A\} \cup \{\neg a'(x) \mid a(x) \in A\}$ is a consistent answer set of π if A' is consistent.

We now prove that A' is consistent. It is easy to see that we only need to prove that A does not contain $holds\_by\_default(l)$ and $holds\_by\_default'(l)$ simultaneously.

Consider a literal $holds\_by\_default(l) \in A$. Since A is a stable model of π_1', there exists rule r of π_1' such that $head(r) = holds\_by\_default(l)$ and $body(r)$ is satisfied in A. There are two cases:

(a) $holds\_by\_default(l) \leftarrow$ is a rule in π_1'. Then, $holds\_by\_default'(l) \leftarrow$ is not a rule in π_1' because X is consistent. Furthermore, for every rule (15.2), r, if $head(r) = holds\_by\_default'(l)$, then $body(r)$ cannot satisfied in A. Hence, $holds\_by\_default'(l) \notin A$.

(b) there exists a rule r of the form (15.2) whose head is $holds\_by\_default(l)$ and whose body is satisfied in A. That implies that $holds\_by\_default'(l) \notin A$ too.

Similar argument can be proven for $holds\_by\_default'(l) \in A$. Therefore, we can conclude that A' is a consistent answer set of π_1. Let $W_{\delta+1} = A'$.

Since $W_{\delta+1}$ contains only literal l of π with $\beta(l) = \delta + 1$ and $\bigcup_{\eta \leq \delta} W_\eta$ contains only literal l with $\beta(l) \leq \delta$ we can conclude that $\bigcup_{\eta \leq \delta+1} X_\eta$ is consistent. The inductive hypothesis is proved. (**)

Let $W_\delta = \emptyset$ for every limit ordinal $\delta < \mu$. From (*) and (**) we can conclude that $\langle W_\delta \rangle_{\delta<\mu}$ is a consistent solution of π with respect to $\langle S_\delta \rangle_{\delta<\mu}$. The lemma is proved. □