Nonmonotonic Reasoning with Well Founded Semantics

Luís Moniz Pereira
AI Centre, Uninova and DCS, U. Nova de Lisboa
2825 Monte da Caparica, Portugal
lmp@fct.unl.pt

Joaquim Nunes Aparício
AI Centre, Uninova and DCS, U. Nova de Lisboa
2825 Monte da Caparica, Portugal
jna@fct.unl.pt

José Júlio Alferes
AI Centre, Uninova and DCS, U. Nova de Lisboa
2825 Monte da Caparica, Portugal
jja@fct.unl.pt

Abstract

Well Founded Semantics is adequate to capture nonmonotonic reasoning if we interpret the Well Founded model of a program P as a (possibly incomplete) view of the world. Thus the Well Founded model may be accepted to be a definite view of the world and the extended stable models as alternative enlarged consistent belief models an agent may have about the world.

Our purpose is to exhibit a modular systematic method of representing nonmonotonic problems with the Well Founded semantics of logic programs. In this paper we use this method to represent and solve some classical nonmonotonic problems. This leads us to consider our method quite generic.

1 Introduction

Well Founded Semantics (WFS) [15] is adequate to capture nonmonotonic reasoning if we interpret the Well Founded model (WFM) of a program P as a (possibly incomplete) view of the world. Thus the WFM may be accepted to be a definite view of the world and the eXtended Stable Models (XSMs) as alternative enlarged consistent belief models an agent may have of the world, all of each containing the WFM. The agent's world view may be completed (or refined) by hypothesizing (e.g. abducing, using default rules, the closed world assumption, etc.) about unknown information. Przymusinski [11, 12, 13] shows that Well Founded semantics is equivalent to suitable forms of four major nonmonotonic reasoning formalisms (Circumscription, Closed World Assumption, Autoepistemic Logic and Default Logic).

In the case the WFM is a 2-valued model then no degree of freedom is

left to the agent to conjecture or hypothesize about the world. If the WFM is 3-valued, hypothetical reasoning in the agent's mind is delimited by the unknown facts of the WFM.

Our purpose is to exhibit a modular systematic method of representing nonmonotonic problems with the Well Founded semantics of logic programs. In this paper we present this method and use it to represent and solve some classical nonmonotonic problems. The programs and some others selected from [5] were tested with a WF Semantic interpreter produced in [7], and the desired conclusions were obtained. This encourages us to consider our method quite generic.

We begin by proposing a representation of information in hierarchical taxonomy problems. In the section "Possible Worlds" we represent hypothetical reasoning problems, interpret the results and propose a representation for unknown knowledge. Then, grounded on the former examples, we present a systematization of our problem representation approach. Afterwards, we use our methodology to represent additional classical nonmonotonic problems (planning, event calculus), arguing that it is sufficiently generic. Finally we stir some discussion and compare the method to related work.

2 Hierarchy Representation

In this section we illustrate how to represent a hierarchy in extended logic programs with the Well Founded semantics. In this representation we wish to express general rules, and their exceptions, as well as exceptions to exceptions. For instance, we want to represent general rules such as "birds normally fly" and "penguins normally don't fly" where each can act as an exception to the other (in the absence of additional information). Furthermore, as in this instance, we wish to be able to express preference for one rule over another in case they conflict and are both applicable. Thus we can prefer the most specific information (e.g. for a penguin, which is a bird we want to conclude that it doesn't fly, unless there is even more specific information regarding possible exceptions to the "penguins normally don't fly" rule).

Our presentation is made step by step using the well known "Birds Fly" hierarchical taxonomy.

2.1 One level hierarchy

Let us begin with the simplest version of this problem: Normally birds fly.

We need a rule $flies(X) \leftarrow bird(X)$ that applies whenever possible, but which can be defeated by exceptions. These exceptions can be had by explicit exception to the conclusion predicate (flies) or by a weaker exception to this rule only. In other words, we want to be able to say that a given bird doesn't fly and also that for a given bird this rule doesn't apply.

In order to allow exceptions to the conclusions, we state that a bird flies only if there is no evidence that it doesn't. Specifically[1]:

$$flies(X) \leftarrow bird(X), \sim\neg flies(X)^{2}.$$

To allow exceptions to the rule, we "name" that rule[3]:

$$flies(X) \leftarrow bird(X), \sim\neg flies(X), bird\_flies(X). \tag{1}$$

To capture the notion that the rule applies whenever possible, i.e. that is a default rule, we introduce:

$$bird\_flies(X) \leftarrow \sim\neg bird\_flies(X). \tag{2}$$

which with (1) captures the notion that if there is no way to prove that an object is an exception to the rule, then assume that the rule applies.

$\neg bird\_flies(X)$ accounts for exceptions to $bird\_flies(X)$. By renaming $\neg bird\_flies(X)$ to McCarthy's $abnormal\_bird(X)$ [6], then from (1) and (2) there follow the rules:

$$\begin{aligned} flies(X) &\leftarrow bird(X), \sim abnormal\_bird(X) \\ abnormal\_bird(X) &\leftarrow \neg flies(X). \end{aligned}$$

We prefer having rule (2), it being a modular statement of an assumption about a predicate. Note also that (2) can be seen as a closed world assumption about $\neg bird\_flies(X)$. Abridging, we represent the initial "Birds Fly" taxonomy with the (extended) logic program:

$$\begin{aligned} \Pi_1 : \quad f(X) &\leftarrow b(X), \sim\neg f(X), bf(X) \\ bf(X) &\leftarrow \sim\neg bf(X) \end{aligned}$$

where f stands for $flies$, b for $bird$ and bf for $bird\_flies$.

Let us see what are the Extended Stable Models (XSM) of $_1$ if we add to it the facts, $Facts_1 = \{b(a), b(b), \neg f(b)\}$. $\Pi_1 \cup Facts_1$ has a unique XS Model coinciding with the Well Founded Model (WFM), which is

$$\{f(a), \sim\neg f(a), \neg f(b), \sim f(b), b(a), b(b), bf(a), \sim\neg bf(a), bf(b), \sim\neg bf(b)\}$$

The fact that this model is 2-valued[4] reflects that no choices are available to obtain other XSMs. This seems quite an acceptable result. Regarding b we know for sure that it is a bird and it doesn't fly. We can say that the rule might apply, because there are no exceptions defined for it. About a we know that it is a bird, and as there is no exception for flies, when X is bound to a, we can say for sure that it flies. Thus the birds fly rule is applied maximally, as default rules are supposed to. This problem domain can next be extended by saying that all penguins are birds, penguins don't fly, and b is a penguin.

2.2 Exceptions to exceptions

In a hierarchy, instead of saying that "penguins don't fly", we would like to say that *normally* penguins don't fly. This will allow us to express an exception to this exception rule for birds fly, and hence the possibility that an exceptional penguin may fly. Let the problem statement now be:

Normally birds fly. Penguins are birds. Normally penguins don't fly.

According to what was said before this problem is represented as:

$$
\begin{array}{lrcll}
{}_2: & f(X) & \leftarrow & b(X), {\sim}\neg f(X), bf(X). & \text{Normally birds fly} \\
 & bf(X) & \leftarrow & {\sim}\neg bf(X). & \text{This is a default} \\
 & \neg f(X) & \leftarrow & p(X), {\sim}f(X), pnf(X). & \text{Normally penguins don't fly} \\
 & pnf(X) & \leftarrow & {\sim}\neg pnf(X) & \text{This is a default} \\
 & b(X) & \leftarrow & p(X). & \text{Penguins are birds}
\end{array}
$$

where p stands for *penguin* and pnf stands for *penguin_not_flies*.

With this program, we consider two birds, one of them being a penguin, $Facts_2 = \{b(a), p(b)\}$. The WF Model of ${}_2 \cup Facts_2$ is:

$$
\begin{array}{l}
WF({}_2 \cup Facts_2) = \\
\quad \{f(a), {\sim}\neg f(a), bf(a), {\sim}\neg bf(a), pnf(a), {\sim}\neg pnf(a), b(a), {\sim}p(a), \\
\qquad\qquad bf(b), {\sim}\neg bf(b), pnf(b), {\sim}\neg pnf(b), b(b), p(b)\}
\end{array}
$$

About a everything is defined, and we have exactly the same case as before. About b this model only tells us that it is a bird and a penguin, leaving the predications $flies$ and $\neg flies$ undefined for it. This is due to the fact that there are two exception rules that might apply, each of which inhibits the conclusion of the other. So nothing can be said for sure about b flying or not.

But the program has two more XS Models:

$$
\begin{array}{l}
XSM_1 = WFM \cup \{f(b), {\sim}\neg f(b)\} \\
XSM_2 = WFM \cup \{\neg f(b), {\sim}f(b)\}
\end{array}
$$

These models can be seen as different models (i.e. consistent beliefs) an agent may have of the world. This is quite an intuitive result, since we can believe that b flies (because it is a bird) or that it doesn't (because it is a penguin). The WF Model is the intersection of these two, and intuitively gives us our definite belief about the world, the things with are able to believe unconditionally. In WFM nothing can be said definitely about b flying or not. This topic will be further discussed in the sequel.

In a hierarchy, however, since we always want to apply the most specific information, there should be in this case an explicit preference between the rules *bird_flies* and *penguin_not_flies*, so as to establish that the property of flying is either true or false of any individual in the taxonomy.

By having rules named, this preference is quite easy to represent. What we wish to say is that if we can apply the rule *penguin_not_flies* to a penguin then we are in presence of an exception to the *bird_flies* rule. So the preference rule is simply:

$$\neg bf(X) \leftarrow p(X), pnf(X)^5.$$

With this rule the only XS Model is the WF Model:

$$\begin{aligned} WF = \{ \quad & f(a), \sim\neg f(a), bf(a), \sim\neg bf(a), pnf(a), \sim\neg pnf(a), b(a), \sim p(a), \\ & \neg f(b), \sim f(b), \neg bf(b), \sim bf(b), pnf(b), \sim\neg pnf(b), b(b), p(b)\} \end{aligned}$$

which expresses our intuitive notion about the hierarchy. b doesn't fly and it is an exception to the *bird_flies* rule.

Next we can represent exceptions to exceptions:

$$\begin{array}{lrcll} Facts_3: & f(X) & \leftarrow & fp(X). & \text{Flying penguins definitely fly} \\ & p(X) & \leftarrow & fp(X). & \text{Flying penguins are penguins} \\ & fp(c). & & & \text{c is a flying penguin} \end{array}$$

where fp is short for *flying_penguin*. The WF Model of ${}_2 \cup Facts_3$ is:

$$WFM = \{f(c), \sim\neg f(c), \neg bf(c), \sim bf(c), pnf(c), \sim\neg pnf(c), b(c), p(c), fp(c)\}$$

Note c is also an exception to the *bird_flies* rule since its a penguin. Nevertheless it flies, but because of the more specific and absolute rule that flying penguins fly.

3 Possible Worlds

In hierarchies, as seen, everything is defined, leaving no choices available. This is not the case for general hypothetical reasoning problems. In this section we represent hypothetical reasoning problems in the Well Founded Models Semantics and interpret the results.

3.1 The "Nixon diamond" problem

Normally quakers are pacifists.	Normally republicans are hawks.
Pacifists are non hawks.	Hawks are non pacifists.
Nixon is a quaker and a republican.	Pacifists are non hawks.
There are other republicans.	There are other quakers.

Using the representation method of last section this example is expressed as:

$$
\begin{array}{rcll}
 & & \text{Normally quakers are pacifists} \\
pacifist(X) & \leftarrow & quaker(X), quaker\_pacifist(X), & \\
 & & \sim\neg pacifist(X). & \\
quaker\_pacifist(X) & \leftarrow & \sim\neg quaker\_pacifist(X). & \\
 & & & \\
 & & \text{Normally republicans are hawks} \\
hawk(X) & \leftarrow & republican(X), republican\_hawk(X), & \\
 & & \sim\neg hawk(X). & \\
republican\_hawk(X) & \leftarrow & \sim\neg republican\_hawk(X). & \\
 & & & \\
\neg hawk(X) & \leftarrow & pacifist(X). & \text{Pacifists are non hawks} \\
\neg pacifist(X) & \leftarrow & hawk(X). & \text{Hawks are non pacifists}
\end{array}
$$

$$
\begin{array}{rl}
quaker(nixon). & \text{Nixon is quaker.} \\
republican(nixon). & \text{Nixon is republican.} \\
quaker(other\_quaker). & \text{There are other quakers.} \\
republican(other\_republican). & \text{There are other republicans}[6]
\end{array}
$$

Here there is no preference defined between the rules nor between their conclusions. So for nixon we want to be able to hypothesize him to be a pacifist or a hawk. Let us have a closer look at the extended stable models of this program, and interpret them carefully. The WFM is[7]:

$$
\begin{array}{l}
\{qua(n), rep(n), qp(n), rh(n), \\
qua(o\_q), qp(o\_q), rh(o\_q), pac(o\_q), \neg hawk(o\_q), \sim\neg pac(o\_q), \sim hawk(o\_q), \\
rep(o\_r), qp(o\_r), rh(o\_r), hawk(o\_r), \neg pac(o\_r), \sim\neg hawk(o\_q), \sim pac(o\_q)\}
\end{array}
$$

For nixon, as expected, it is unknown whether he is a pacifist or a hawk. Nevertheless we have quaker_pacifist(nixon) and republican_hawk(nixon), so that both rules applied. This is not a strange result since rules are maximally applicable, and nixon does not consist an exception to the rules through to their conclusions. Of course, for other quakers and other republicans everything is defined.

Since we have unknown literals not present in the WF Model we might have other extended stable models. In this case the other XS Models are[8]:

$$
\begin{array}{l}
XSM_1 = WFM \cup \{pac(n), \neg hawk(n)\} \\
XSM_2 = WFM \cup \{\neg pac(n), hawk(n)\}
\end{array}
$$

These remaining XS Models can be seen as possible extended world views in which some consistent choices of belief have been made. XSM_1 represents a world view where nixon is a pacifist and a non hawk, and XSM_2 represents a world view where nixon is a hawk and a non pacifist.

3.2 Unknown application of rules

In the last example, exceptions are not present on the application of rules, but only on their conclusions, and so the rules were maximally applied. But in some cases we would like to choose between applying or not some rule, by keeping the rule name unknown in the WF Model.

Above, a rule of the form $name\_of\_rule(X) \leftarrow \sim\neg name\_of\_rule(X)$ expresses that the rule is true if possible. To have the possibility of applying or not the rule, we add a clause stating that the rule name is not true if possible, gaining this way freedom to apply or not the rule. The additional clause is clearly:

$$\neg name\_of\_rule(X) \leftarrow \sim name\_of\_rule(X).$$

With this formulation rule names, and consequently their conclusions, become unknown in the WF Model, if nothing else is said about their name. XS Models appear: ones with the rule name being true, and others with it being false.

Furthermore, one might want to have the possibility of applying a rule or not, but only under certain conditions, and applying it maximally otherwise. As an example suppose:

Normally quakers are pacifists. (1)	Nixon is a quaker.
We are not sure if (1) applies to nixon. (2)	There other quakers.

Because of (2), we wish to choose between having or not rule (1) applied to nixon, but to apply it maximally to other individuals. Given (2), rule (1) becomes a rule with a possibly undecided applicability.

Accordingdly, our representation of this problem is:

$$\begin{array}{rcl}
 & & \text{Normally quakers are pacifists} \\
pacifist(X) & \leftarrow & quaker(X), quaker_pacifist(X), \\
 & & \sim\neg pacifist(X). \\
quaker_pacifist(X) & \leftarrow & \sim\neg quaker_pacifist(X). \\
 & & \\
 & & \text{Rule (1) might not apply to nixon} \\
\neg quaker_pacifist(nixon) & \leftarrow & \sim quaker_pacifist(nixon). \\
quaker(nixon). & & \text{Nixon is a quaker.} \\
quaker(other_quaker). & & \text{There are other quakers.}
\end{array}$$

The WF Model of this program is:

$$\begin{array}{rl}
WFM = \{ & quaker(nixon), quaker(other_quaker), \\
 & pacifist(other_quaker), \sim\neg pacifist(other_quaker)\}
\end{array}$$

and its consistent XS Models are[9]:

$$\begin{array}{rl}
XSM_1 = & WFM \cup \{pacifist(nixon), quaker_pacifist(nixon)\} \\
XSM_2 = & WFM \cup \{\neg quaker_pacifist(nixon)\}
\end{array}$$

In the WF Model we are not able to conclude anything about nixon being a pacifist because we are not able to apply the rule.

XSM_1 represents a world where nixon is a pacifist, by choosing to apply the rule quaker_pacifist via its name being true; XSM_2 is a world where we choose not to apply the corresponding rule, via its name being false, so that we can prove nothing about nixon being a pacifist[10].

The interpretation of the above results, in particular in what concerns XSMs where literals not in the WF Model become true, suggested to us a relationship between the XS Models of a program with rules like (1) and the concept of abduction within WF Semantics [9].

3.3 Unknown possible facts

Similarly to rules about which we are undecided regarding their applicability, we might be unsure about some facts. Note that this is different from not having any knowledge at all about such a fact. Consider this simple example:

John and Nixon are quakers. John is a pacifist.

represented by the program:

$$\Pi = \{quaker(john), quaker(nixon), pacifist(john)\}.$$

The WFM(which is the only XS Model) is:

$$\{quaker(john), quaker(nixon), pacifist(john), \sim pacifist(nixon)\}$$

and expresses exactly what is intended, i.e. john and nixon are quakers, john is a pacifist and we don't have reason to believe nixon is a pacifist. Now suppose we add: Nixon might be a pacifist (3). In our view, we wouldn't want in this case to be so strong as to affirm $\sim pacifist(nixon)$, thereby not allowing for the possibility of nixon being a pacifist. What we are prepared to say is that Nixon might be a pacifist if we don't have reason to believe he isn't and, vice-versa, that Nixon might be a non pacifist if we don't have reason to believe he isn't one. This is clearly expressed as:

$$\begin{array}{rcl} pacifist(nixon) & \leftarrow & \sim\neg pacifist(nixon). \\ \neg pacifist(nixon) & \leftarrow & \sim pacifist(nixon). \end{array}$$

Program together with these rules has

$$WFM = \{quaker(john), quaker(nixon), pacifist(john), \sim\neg pacifist(john)\},$$

and two more XS Models:

$$\begin{array}{l} XSM_1 = WF \cup \{pacifist(nixon), \sim\neg pacifist(nixon)\} \\ XSM_2 = WF \cup \{\neg pacifist(nixon), \sim pacifist(nixon)\} \end{array}$$

which is the result we were seeking. Facts like (3) are what we call *unknown possible facts.*

4 Summary of our representation method

In this section we summarize and systematize the representation method adopted in all the above examples. The type of rules for which we propose a representation is, in our view, general enough to capture a wide domain of nonmonotonic problems. Each type of rule is described in a subsection by means of a schema in natural language and its corresponding representation schema .

- **Definite Rules** *If A then B.* The representation is: $B \leftarrow A$.
- **Definite Facts** *A is true.* The representation is: A.
- **Default (or maximally applicable) Rules** *Normally if A then B.* The representation is:

$$\begin{aligned} B &\leftarrow A, name\_A\_B, \sim\neg B. \\ name\_A\_B &\leftarrow \sim\neg name\_A\_B. \end{aligned}$$

  where $name\_A\_B$ is a predicate symbol that "names" this rule only. Its arguments are those arguments in A or B. We will consider *Default Facts* as a special case of *Default Rules* where A is absent. As an example consider the rule "Normally birds fly". Its representation is:

$$\begin{aligned} fly(X) &\leftarrow bird(X), bird\_flies(X), \sim\neg fly(X). \\ bird\_flies(X) &\leftarrow \sim\neg bird\_flies(X). \end{aligned}$$

- **Possible Rules** *Rule "If A then B" may or may not apply.* Its representation is:

$$\begin{aligned} B &\leftarrow A, name\_A\_B, \sim\neg B. \\ name\_A\_B &\leftarrow \sim\neg name\_A\_B. \\ \neg name\_A\_B &\leftarrow \sim name\_A\_B. \end{aligned}$$

  where name_A_B is a predicate symbol that "names" the first rule only. Its arguments are all those arguments in A or B. As an example consider the rule "Quakers might be pacifists". Its representation is:

$$pacifist(X) \leftarrow quaker(X), quaker\_pacifist(X), \sim\neg pacifist(X).$$

$$\begin{aligned} quaker\_pacifist(X) &\leftarrow \sim\neg quaker\_pacifist(X). \\ \neg quaker\_pacifist(X) &\leftarrow \sim quaker\_pacifist(X). \end{aligned}$$

- **Exceptions to Default Rule** *Under certain conditions COND there are exceptions to the default rule named NAME.*

$$\neg NAME \leftarrow COND.$$

  As an example, the representation of the exception "Penguins are exceptions to the "normally birds fly" rule" is:

$$\neg bird\_flies(X) \leftarrow penguin(X).$$

- **Possible Exceptions to Default Rule** *Under certain conditions COND there might be exceptions to the default rule named NAME.*

$$\neg NAME \leftarrow COND, \sim NAME.$$

As an example, the representation of the exception "Nixon is a possible exception to the "normally quakers are pacifists" rule" is:

$$\neg quaker\_pacifist(nixon) \leftarrow \sim quaker\_pacifist(nixon).$$

- **Preference Rules** *Under conditions COND, prefer to apply the default rule NAME+ instead of the default rule named NAME-.*

$$\neg NAME- \leftarrow COND, NAME+.$$

As an example consider "For penguins, if the rule that says that "normally penguins don't fly" is applicable then inhibit the "normally birds fly" rule". This is represented as:

$$\neg bird\_flies(X) \leftarrow penguin(X), penguins\_no\_flies(X).$$

- **Unknown Possible Fact** *F might be true or not* (in other words, the possibility of F should be considered).

$$\begin{aligned} F &\leftarrow \sim\neg F. \\ \neg F &\leftarrow \sim F. \end{aligned}$$

5 Application To Other Problems

We now apply the programing method described above to some other problems. We will not argue that this method is the best for representing these problems. Instead, we want to show that it gives the right results, so being a general method to represent nonmonotonic problems.

5.1 The Yale Shooting Problem

This problem, supplied in [3], will be represented in a form near to the one suggested in [4]. Predicate $holds(P, S)$ expresses that property P holds in situation S; predicate $normal(P, E, S)$ expresses that in situation S, event E does not normally affect the truth value of property P; the term $result(E, S)$ names the situation resulting from the occurrence of event E in situation S.

The problem and its formulation are as follows:

Initially (in situation s0) a person is alive: $holds(alive, s0)$.

After loading a gun the gun is loaded: $holds(loaded, result(load, S))$.

If the gun is loaded, then after shooting it the person will not be alive. Note that the latter is not a normally-rule:

$$\neg holds(alive, result(shoot, S)) \leftarrow holds(loaded, S).$$

After an event things normally remain as they were, i.e. properties which hold before will normally still hold after the event; likewise, properties which do not hold before the event will normally not hold afterwards as well. Here predicate $normal(P, E, S)$ will be used to name both these rules:

$holds(P,result(E,S)) \leftarrow$
$\quad holds(P,S), normal(P,E,S), \sim\neg holds(P,result(E,S))$ (pp)[11]
$\neg holds(P,result(E,S)) \leftarrow$
$\quad \neg holds(P,S), normal(\neg P,E,S), \sim holds(P,result(E,S))$ (np)[12]
$normal(P,E,S) \leftarrow$
$\quad \sim\neg normal(P,E,S)$

In principle we should use different names for each of the rules. In fact $normal(\neg P, E, S)$ is just a syntactic gimmick to avoid a new name predicate for (np), and thus avoid an additional rule for it.

Consider the question "What holds and what doesn't hold after the loading of a gun, a period of waiting and a shooting ?" represented as two queries:

$$\leftarrow holds(P, result(shoot, result(wait, result(load, s0))))$$
$$\leftarrow \neg holds(P, result(shoot, result(wait, result(load, s0))))$$

With this formulation the WF Model is the only XS Model. The subset of its elements that match with at least one of the queries is[13]:

$$\{holds(loaded, s3), \sim\neg holds(loaded, s3), \neg holds(alive, s3), \sim holds(alive, s3)\}$$

which means that in situation s3 the gun is loaded and the person is not alive. This result coincides with the one obtained in [4] for holds.

To get the result given by circumscription [6] and default logic [14], we must reformulate the problem by adding the sentence: the *wait* event might not preserve the persistence of the *loaded* property; in other words, after a *wait* event the gun might be unloaded. This clearly requires an unknown application of (pp). So the rule to add is:

$$\neg normal(loaded, wait, S) \leftarrow \sim normal(loaded, wait, S).$$

Now the subset of the WF Model is $\{\sim\neg holds(loaded, s3)\}$. This means that in the WFM we have no proof that the gun is not loaded. This is acceptable because there is no evidence for it to be unloaded. All other properties are unknown in the WF model.But we have two XS Models, corresponding to the two default extensions. Their subsets of elements that match a query are:

$$\{holds(alive, s3), \sim\neg holds(alive, s3), \sim\neg holds(loaded, s3)\}$$
$$\{\neg holds(alive, s3), \sim holds(alive, s3), holds(loaded, s3), \sim\neg holds(loaded, s3)\}$$

We tested this example with other modifications, for example (1) Considering that having a gun loaded after a wait action is an exception to the (pp) rule. (2) In the initial situation the gun might be loaded.

In applications to some other problems, in an extended version of this paper, our representation method also gives quite intuitive results.

6 Discussion and open work

Here we raise points of discussion, including comparisons with other work.

6.1 Using contrapositives of default rules

One might wish to have contrapositive variants of a default rule. For example, one might want it holds to, given the "normally birds fly" rule, that an individual which doesn't fly, is normally not a bird.

Given the "birds fly" rule:

$$flies(X) \leftarrow bird(X), bird\_flies(X), \sim\neg flies(X) \ (c1)$$

its sole contrapositive is:

$$\neg bird(X) \leftarrow \neg flies(X), bird\_flies(X), \sim bird(X) \ (c2)$$

This has the disadvantage of using the same rule name in both, and being somewhat repetitive. Syntactically, one could write rules allowing contrapositives in a distinct form, for example: $(flies(X) \Leftarrow bird(X)) \leftarrow bird\_flies(X)$, which, of course would stand for (c1) and (c2), and where $\Leftarrow$ stands for material implication.

Note that the last literal of each of the above rules, is needed only to avoid contradictory well founded models.

For example, consider the program $P = \{f \leftarrow b, bf \quad bf \leftarrow\sim\neg bf \quad b \quad \neg f\}$ not having such literals. This program has a contradictory WF Model. However, adding $\sim\neg f$ to the body of the first clause, leads to a noncontradictory WF Model containing $\neg f$. The effect of those literals in the WF Model can be seen as a preference for the use of definite rules over default ones. The next point of discussion suggests a modification to the semantics in order to have that preference without the need for those literals.

6.2 Avoiding contradiction in the WF Model

A literal assumed false[14] in the WF Model (Closed World Assumption (CWA)), can lead to a contradictory WFM when classical negation is present. If an assumption leads to a contradiction it seems natural to be able to go back on that assumption.

A suggestion is to build a new class of models, defined as being the noncontradictory models, resulting from allowing some literals assumed false by CWA in the WF Model to have the unknown truth value, just in case the WF Model is of contradictory. All these models would be smaller, by Fitting's ordering [1], than the WFM. For this reason we call them Sub Well Founded Models (SWF Models)[8]. The maximal SWF Models can also provide Sub XSM models.

Note that for programs having a contradiction derived from definite rules or facts (Ex: $\{p.\ \neg p.\}$) there are no SWF Models. As these models remove

contradiction by changing the truth value only on NAF negated literals, and those literals never appear in definite rules, they reflect preference on the use of these rules over the use of default ones, as proposed above.

6.3 Comparison with "Logic Programs With Exceptions"[4]

We deal with exceptions to exceptions in a uniform and modular way. Because of its inherent asymmetry, the "rules with exceptions" approach requires changing previous rules in the program each time an exception to an exception is made, because head literals need to change. For instance, a three level hierarchy of birds, penguins and flying penguins requires rules like

$$\begin{aligned} fly(X) &\leftarrow bird(X) \\ nofly(X) &\leftarrow penguin(X) \\ fly(X) &\leftarrow flying\_penguin(X) \end{aligned}$$

and the exceptions:

$$\begin{aligned} \neg fly(X) &\leftarrow nofly(X) \\ \neg nofly(X) &\leftarrow flying\_penguin(X) \end{aligned}$$

We allow both positive and negative conclusions in rules, inclusively for the same predicate.

The extension of well founded semantics to classical negation provides a (non contradictory) well founded model of definite conclusions in cases where e-answer set semantics provides only alternative models. For instance, in the pacifist/hawk example we obtain a well founded model containing the facts {quaker, republican}, besides the two alternative e-answer sets. By introducing "name predicates" in rules, we have shown how to express exceptions to rules, rather than exceptions to whole predicates, and how to express preference amongst rules within the language.

6.4 Comparison with "Answer-set Semantics"[2]

The three-valued semantics subsumes the two-valued semantics of answer-set semantics. The techniques for nonmonotonic reasoning we've presented are in part adoptable by the answer-set semantics of extended logic programs.

6.5 Comparison with "Poole's naming device"[10]

We deal with preference between rules within the language. Poole does this using his constraints. These constraints are not in the language of logic programs, and are in fact a meta-level model-theoretic construct outside it.

The use of formulas by Poole, instead of rules, includes all contrapositives even if not wanted. Again the pruning effect can only be achieved by the use of constraints. In our method we have only the contrapositives effectively written. If all are wanted all can be included.

Acknowledgements

We thank ESPRIT Basic Research Project COMPULOG (no. 3012), Instituto Nacional de Investigação Científica and Gabinete de Filosofia do Conhecimento for their support. Thanks to Fariba Sadri and Tony Kakas for fruitful discussions.

Notes

[1]This rule is similar to Reiter's normal defaults [14], where bird(X) is the precondition

[2]The symbol ∼stands for the negation as failure operator. ¬ stands for classical negation as in [2]

[3]This is akeen to Poole's naming device [10], but used for rules instead of formulas

[4]Note that all predicates and their classical negations are defined

[5]Poole [10] does this using his constraints, a meta-level model-theoretic construct outside the language of logic programs

[6]other_quaker and other_republican can be envisaged as Skolem constants

[7]Where qua stands for quaker, rep for republican, pac for pacifist, qp for quaker_pacifist, rh for republican_hawk, n for nixon, o_q for other_quaker and o_r for other_republican

[8]As these two models are 2-valued we don't show the additional ∼literals, which are implicit

[9]Again, as the two models are 2-valued we don't present the additional ∼ literals

[10]Note that this model contains $\sim pacifist(nixon)$ and $\sim\neg pacifist(nixon)$

[11]pp stands for positive persistence

[12]np stands for negative persistence

[13]Where s3 denotes the term result(shoot,result(wait,result(load,s0))).

[14]A literal is assumed false in the WFM if it appears under ∼there.

References

[1] M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[2] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *ICLP90 -Israel*, pages 579–597, September 1990.

[3] S. Hanks and D. McDermott. Default reasoning, nonmonotonic logics and the frame problem. In *AAAI-86*, pages 328–333, August 1986.

[4] R. Kowalski and F. Sadri. Logic programs with exceptions. In *ICLP90 -Jerusalem. Israel*, 1990.

[5] V. Lifschitz. Benchmarks problems for formal nonmonotonic reasoning. In M. Reinfrank, J. d. Kleer, M. Ginsberg, and E. Sandewall, editors, *Non Monotonic Reasoning: 2nd International Workshop -Grassau, FRG*, pages 202–219. Springer Verlag, 1988.

[6] J. McCarthy. *Applications of Circumscription to Formalizing Common-Sense Knowledge*. Readings In Nonmonotonic Reasoning. Morgan Kaufman Publishers, Inc, 1987. Also in AI journal v28 (1986).

[7] L. M. Pereira, J. N. Aparício and J. J. Alferes . A derivation procedure for extended stable models. Technical report, CRIA/UNINOVA, December 1990.

[8] L. M. Pereira, J. J. Alferes, and J. N. Aparício. Contradiction within Well Founded Semantics. Technical report, CRIA/UNINOVA, 1991.

[9] L. M. Pereira, J. N. Aparício, and J. J. Alferes. Hypothetical reasoning with well founded semantics. Technical report, CRIA/UNINOVA, 1990.

[10] D. L. Poole. A logical framework for default reasoning. *Artificial Intelligence*, 36(1):27–47, 1988.

[11] T. C. Przymusinski. On the relantionship between logic programming and non-monotonic reasoning. In *AAAI88 -Saint Paul, Minnesota*, pages 444–448. Morgan Kaufmann, 1988.

[12] T. C. Przymusinski. Non-monotonic formalisms and logic programming. In *ICLP-89 -Lisbon, Portugal*, pages 655–674. mit press, 1989.

[13] T. C. Przymusinski. Three-valued formalizations of non-monotonic reasoning and logic programming. In H. Levesque R. Brachman and R. Reiter, editors, *1st Conference on Principles of Knowledge Representation and Reasoning -Toronto, Canada*, pages 341–348. Morgan Kaufmann Publishers, San Mateo, California, 1989.

[14] R. Reiter. *A Logic for default Reasoning*. Readings In Nonmonotonic Reasoning. Morgan Kaufman Publishers, Inc, 1987. Also in AI13(1980).

[15] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of ACM*, pages 221–230, 1990.