Non–monotonic Reasoning with Logic Programming

Luís Moniz Pereira, Joaquim N. Aparício, José J. Alferes
CRIA, Uninova and DCS, U. Nova de Lisboa
2825 Monte da Caparica, Portugal
{lmp|jna|jja}@fct.unl.pt

October 18, 1993

Abstract

Our purpose is to exhibit a modular systematic method of representing non–monotonic reasoning problems with the Well Founded Semantics WFS of extended logic programs augmented with eXplicit negation (WFSX), augmented by its Contradiction Removal Semantics (CRSX) when needed. We apply this semantics, and its contradiction removal semantics counterpart, to represent non-monotonic reasoning problems. We show how to cast in the language of logic programs extended with explicit negation such forms of non-monotonic reasoning as defeasible reasoning, abductive reasoning and hypothetical reasoning and apply them to such different domains of knowledge representation as hierarchies and reasoning about actions. We then abstract a modular systematic method of representing non-monotonic problems in a logic programming semantics comprising two forms of negation avoiding some drawbacks of other proposals, with which we relate our work.

Contents

1 Introduction **4**

2 CRSX Review **6**
2.1 Language Used 6
2.2 WFSX overview 6
2.3 Revising contradictory extended logic programs 9
2.3.1 Contradiction Free Programs 13

3 Summary of our representation method **15**

4 Explicit Negative Information **16**

5 Defeasible Reasoning **17**
5.1 Exceptions 20
5.1.1 Exceptions to predicates 20
5.1.2 Exceptions to rules 21
5.1.3 Exceptions to exceptions 21
5.2 Preferences among rules 22

6 Representation of Hierarchical Taxonomies **22**

7 Hypothetical Reasoning **26**
7.1 The birds world 26
7.2 Hypothetical facts and rules 28
7.2.1 Hypothetical facts 28
7.2.2 Hypothetical rules 30

8 Application to Reasoning about Actions **33**
8.1 The Yale Shooting Problem 33
8.2 Multiple Extensions 34
8.3 Other Reasoning About Action Problems 34

9 Abduction within CRSX **36**

10 Related work **37**

A CRSX Review **44**

1. Introduction

Recently, several authors have stressed and showed the importance of having an explicit second kind of negation within logic programs, for use in deductive databases, knowledge representation, and non–monotonic reasoning [3, 6, 11, 18, 16, 32, 33, 34, 43, 39].

In non–monotonic reasoning with logic programming there are two main ways of giving meaning to sets of rules when a given semantics is assigned to a program defined by the set of rules. We either accept as consequences the intersection of all models identified by some semantics, which is called *skeptical reasoning* [21, 2], or we consider one particular model identifying the consequences of a given set of assumptions - this form of reasoning is called *brave* reasoning in [21].

It has been argued [33, 34, 32, 31] that semantics with the well founded property are adequate to capture non–monotonic reasoning if we interpret the least model provided by the semantics (called the Well Founded Model) as the skeptical view of the world and the other models (called Extended Stable Models) as alternative enlarged consistent belief sets standing for different possibilities of brave reasoning. A consequence of the well founded property is that intersection of all models identified by the semantics is itself a model belonging to the semantics. Thus proof procedures for capturing skeptical reasoning may be related to one model in the semantics – or equivalently to validity in all models – thus properly recasting the classic logical notion of entailment in logic programming. This is the case with Przymusinski's Extended Stable Model semantics [37].

Some proposals for extending logic programming semantics with a second kind of negation has been proposed. One such extension is the Answer Set semantics [6], which is shown to be an extension of Stable Model (SM) semantics [5] from the class of logic programs [20] to the class of logic programs with a second form of negation. In [18] another proposal for such extension is introduced, based on the SM semantics, where implicitly a preference for negative information (exceptions) over positive information is assumed. However SM semantics is not well founded and even if the meaning of the program is defined as the intersection of all stable models, it is known that the computation of this intersection is computationally expensive. Another extension to include a second kind of negation is suggested by Przymusinski in [38]. Although the set of models identified by this extension enjoys the well founded property, it gives some less intuitive results [1] with respect to the coexistence of both forms of negation. Based on the XSM semantics Przymusinski [39] also introduces the Stationary semantics where the second form of negation is classical negation. Unfortunately, classical negation also implies that the logic programs under Stationary semantics no longer admit a procedural reading.

Well Founded Semantics with Explicit Negation (WFSX) [26] is an extension to Well Founded Semantics [42] (WFS) including a second form of negation called *explicit*

negation, preserving the well founded property. Furthermore, *explicit negation* is characterized by that, in any model, whatever the classical literal l, $l \wedge \neg l$ never holds, and by that whenever $\neg l$ holds $\sim l$, the negation by default or implicit negation of l also holds, and l is false, thus avoiding the less intuitive results concerning the relation between the two forms of negation. However, $l \vee \neg l$ is not mandatory, so the ability is kept for the truth value of some literals to remain undefined (cf. [1] for other approaches).

When a second form of negation is introduced contradiction may be present (i.e. l and $\neg l$ hold for some l) and no semantics is given by WFSX[1]. In [29] the authors define CRSX extending WFSX by introducing the notion of removing some contradictions and identifying the models obtained by revising closed world assumptions supporting those contradictions. One unique model, if any such revised model exists, is singled out as the contradiction free semantics. When no contradiction is present CRSX semantics reduces to WFSX semantics.

Furthermore, under WFSX programs admit a procedural logic programming reading, which is not the case if truly classical negation plus material implication are used, as in [39], where case analysis is condoned. Under WFSX rules in the program are unidirectional (contrapositives are not implicit), maintaining the procedural flavour; the rule connective, $\leftarrow$, is not material implication, but is rather like an inference rule.

Here we show how to cast in the language of logic programs extended with explicit negation different forms of non-monotonic reasoning such as defeasible reasoning, abductive reasoning and hypothetical reasoning, and apply it to diverse domains of knowledge representation such as hierarchies and reasoning about actions.

Our main purpose is to abstract out and exhibit a modular and systematic method of representing non–monotonic reasoning problems with our CRSX semantics of logic programs. We argue that Logic Programming extended with the concept of undefinedness and a suitable form of explicit negation, is very rich to represent such problems.

This paper is organized as follows. In section 2 we review CRSX semantics [29] which is an extension of the (WFSX) [26] for programs which have no WFSX semantics. Then we identify simple forms of commonsense reasoning (e.g. defeasible reasoning with exceptions, hypothetical reasoning) and show how they are represented by logic programs when CRSX is used. Using the notion of defeasibility and exception rules we then show how to formalize hierarchical reasoning where exceptions are also present. Next we represent problems where hypothetical reasoning is used to capture brave reasoning.

Afterwards we use our approach to represent additional classical non–monotonic problems in reasoning about actions, arguing that it is sufficiently generic. Then, grounded on the former examples, we abstract a systematization of our problem representation methodology. Finally we mention and compare with related work.

[1] In [30, 31] it is shown how WFSX relates to default theory.

2. CRSX Review

In this section we review a method for giving meaning to extended logic programs applicable whenever WFSX is taken as the semantics and the program is contradictory. We first review WFSX semantics [26] and next the method for revising contradictory programs [29].

2.1. Language Used

Given a first order language $Lang$, an extended logic program is a set of rules of the form $H \leftarrow B_1, \ldots, B_n, \sim C_1, \ldots, \sim C_m \quad m \geq 0, n \geq 0$, where $H, B_1, \ldots, B_n, C_1, \ldots, C_m$ are classical literals. A (syntactically) classical literal (or explicit literal) is either an atom A or its explicit negation $\neg A$. We use the symbol $\neg$ to denote complementary literals in the sense of explicit negation. Thus $\neg\neg A = A$. The symbol $\sim$ stands for negation by default[2]. $\sim L$ is called a default literal. Literals are either classical or default literals. A set of rules stands for all its ground instances w.r.t. $Lang$. When $(n = m = 0)$ we may simply write H instead of $H \leftarrow$. If S is a set of literals we say S is *contradictory* iff there is an atom L such that $\{L, \neg L\} \subseteq S$. In this case we also say that S *is contradictory w.r.t.* L. If S is a set by $\sim S$ we mean the set $\{\sim L | L \in S\}$.

2.2. WFSX overview

In this section we briefly review WFSX Well Founded Semantics for logic programs extended with eXplicit negation. For full details and the relation with default logic the reader is referred to [26, 30, 31]. WFSX follows from WFS plus one basic "coherence" requirement: $\neg L$ entails $\sim L$ (if L is explicitly false, L must be false) for any literal L.

Example 1. Take program $P = \{a \leftarrow \sim b \ ; b \leftarrow \sim a \ ; \neg a \leftarrow\}$. If $\neg a$ is simply envisaged as a new atom symbol, say a' (as suggested in [38]), and well founded semantics [42] is used to define the semantics of P, the meaning of P is $\{\neg a, \sim\neg b\}$, so that $\neg a$ is true and a is undefined. We insist that $\sim a$ should hold because $\neg a$ does. Accordingly, the WFSX semantics of P is $\{\neg a, b, \sim a, \sim\neg b\}$, since b follows from $\sim a$.

□

Intuitively this can be construed as there existing an inference rule

$$\neg L \Rightarrow \sim L \tag{1}$$

[2] This designation has been used in the literature instead of the more operational *"negation as failure (to prove)"*. Another appropriate designation is *"implicit negation"*, in contradistinction to *explicit negation*. When $\neg A$ is treated as a new predicate symbol in the way as suggested in [38] we call it *pseudo–negation*. A comparison among different two types of negation in logic programming can be found in [1].

stating that whenever $\neg L$ holds $\sim L$ must also hold[3], i.e. if $\neg L$ is true L is false.

Definition 2.1 (Interpretation). By an interpretation I of a language $Lang$ we mean any set $T \cup \sim F$, where T and F are disjoint subsets of classical ground literals over the Herbrand base, and if $\neg L \in T$ then $L \in F$ (coherence)[4]. The set T contains all ground classical literals *true* in I, the set F contains all ground classical literals *false* in I. The truth value of the remaining classical literals is *undefined* (the truth value of a default literal $\sim L$ is the 3-valued complement[5] of L).

To account for coherence we next extend with an additional rule the P modulo I transformation of [36], itself an extension of the Gelfond-Lifschitz modulo transformation:

Definition 2.2 (P/I transformation). Let P be an extended logic program and let I be an interpretation. By P/I we mean a program obtained from P by performing the following four operations, of which the second only is novel:

- Remove all rules containing a default premise $\sim L$ such that $L \in I$.
- Remove all rules with a non–default premise L (resp. $\neg L$) such that $\neg L \in I$ (resp. $L \in I$).
- Remove from all rules their default premises $\sim L$ such that $\sim L \in I$.
- Replace all the remaining default premises by proposition $\mathbf{u}$[6].

The modification introduced is explained as follows: the second rule in the definition has the effect that if I is to be a model containing $\neg L$ then it also must have $\sim L$, by definition of interpretation, and L is false. Since I may not be contradictory it does not have L and any rule with L in the body may thus be discarded. On the other hand there is no need for a rule removing default literals $\sim L$ such that $\neg L \in I$ because either I is coherent (see definition below) and contains $\sim L$ and those literals will be taken into account by the third rule in the definition, or I is not coherent and then it will not be a fixed point.

The resulting program P/I is by definition non–negative, and thus it always has a unique $least(P/I)$ adapted from [36] (cf. its definition in the appendix to this paper).

Note that $least(P)$ is not always an interpretation in the sense of definition 2.1. Conditions about noncontradiction and coherence may be violated.

[3] Recall that since $\neg\neg L \equiv L$, expression (1) also means $L \Rightarrow \sim\neg L$.

[4] For any literal L, if L is explicitly false L must be false. Note that the complementary condition "if $L \in T$ then $\neg L \in F$" is implicit.

[5] The 3-valued complement operation $\sim$ over the set $\{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$ of truth values is defined as $\sim\mathbf{t} = \mathbf{f}$, $\sim\mathbf{f} = \mathbf{t}$ and $\sim\mathbf{u} = \mathbf{u}$.

[6] The special proposition $\mathbf{u}$ is *undefined* in all interpretations.

Example 2. Consider the non-negative program $P = \{\neg a \leftarrow \neg b, \neg b \leftarrow, b \leftarrow \mathbf{u}\}$, where $least(P) = \{\neg a, \neg b\}$. This set is not an interpretation. □

To avoid incoherence, when contradiction is not present, we define a partial operator that transforms any noncontradictory set of literals into an interpretation. This operator is applied to any non–contradictory $least(P/I)$.

Definition 2.3 (Coh operator). Let $I = T \cup \sim F$ be a set of literals such that T is not contradictory. We define $Coh(I) = I \cup \sim\{\neg L \mid L \in T\}$. Coh is not defined for other sets of literals.

Definition 2.4 (Φ operator). Let P be a logic program, I an interpretation, and $J = least(P/I)$. If $Coh(J)$ exists we define $\Phi_P(I) = Coh(J)$. Otherwise $\Phi_P(I)$ is not defined.

Definition 2.5 (WFS with explicit negation). An interpretation I of an extended logic program P is called an Extended Stable Model (XSM) of P iff $\Phi_P(I) = I$. The F-least Extended Stable Model is called the Well Founded Model. The semantics of P is determined by the set of all XSMs of P.

Example 1. (**cont.**)For the given program we have $P/\{a', b, \sim a, \sim b'\} = \{b \leftarrow; a' \leftarrow\}$ and $least(P/\{a', b, \sim a, \sim b'\}) = \{a', b, \sim a, \sim b'\}$. □

Example 3. Let P be:

$$\begin{aligned} a &\leftarrow \sim b, \sim c \\ b &\leftarrow \sim a \\ \neg c &\leftarrow \sim d \end{aligned}$$

This program has a least model $M_1 = \{\sim d, \neg c, \sim c, \sim \neg a, \sim \neg b, \sim \neg d\}$ and two Extended Stable Models $M_2 = M_1 \cup \{\sim a, b\}$ and $M_3 = M_1 \cup \{a, \sim b\}$

Considering model M_1 we have for P/M_1 :

$$\begin{aligned} a &\leftarrow \mathbf{u} \\ b &\leftarrow \mathbf{u} \\ \neg c &\leftarrow \end{aligned}$$

and $least(P/M_1) = J = \{\sim d, c', \sim a', \sim b', \sim d', \sim c\} = \{\sim d, \sim\neg d, \neg c, \sim c, \sim\neg a, \sim\neg b\}$. □

In the examples above the coherence principle was not needed, i.e. $Coh(least(\Phi_P(I))) = least(\Phi_P(I))$. However this is not always the case as shown by the following example:

Example 4. Let P be:

$$\begin{array}{rll} a & \leftarrow\sim a & (i) \\ b & \leftarrow\sim a & (ii) \\ \neg b & \leftarrow & (iii) \end{array}$$

After the transformation, program P' has a rule $b' \leftarrow$, and there is no way in proving $\sim b$ from rules (i) and (ii). And we have $least(P'/\{b', \sim b, \sim a'\}) = \{b', \sim a'\} = M$ which corresponds to the model $\{\neg b, \sim\neg a\}$ if the coherence principle is not applied. In our case we have $Coh(M) = \{\neg b, \sim b, \sim\neg a\}$ which is the intended result. □

Definition 2.6 (Contradictory program). An extended logic program P is contradictory iff it has no semantics, i.e. there exists no interpretation I such that $\Phi_P(I) = I$.

2.3. Revising contradictory extended logic programs

Here we review the semantics defined in [29]. For full details, properties (including those regarding the minimality criterium), and for comparisons with other semantics, the reader is refered to that report.

Once we introduce explicit negation programs are liable to be contradictory:

Example 5. Consider program $P = \{a \leftarrow ; \neg a \leftarrow\sim b\}$. Since we have no clauses for b, by CWA it is natural to accept $\sim b$ as true. By the second rule in P we have $\neg a$, leading to an inconsistency with the fact a. Thus no set containing $\sim b$ may be a model of P. □

We argue that the CWA may not be held of atom b since it leads to a contradiction (reductio ad absurdum). We show below how to revise[7] this form of contradiction, by making a suitable revision of the incorrect CWA on b. This semantics identifies $\{a, \sim\neg a\}$ as the intended meaning of P, where b is revised to undefined. Assuming b false leads to a contradiction; revising it to true instead of to undefined would not minimize the revised interpretation.

In order to revise possible contradictions we need to identify those contradictory sets implied by applications of CWA. The main idea is to compute all consequences of the program, even those leading to contradictions, as well as those arising from contradictions. The following example provides an intuitive preview of what we intend to capture:

Example 6. Consider program P :

$$\begin{array}{rlll} a \leftarrow & \sim b & \text{(i)} & \qquad d \leftarrow \quad a \quad \text{(iii)} \\ \neg a \leftarrow & \sim c & \text{(ii)} & \qquad e \leftarrow \quad \neg a \quad \text{(iv)} \end{array}$$

[7] We treat contradictory programs extending the approach of [27, 28].

1. $\sim b$ and $\sim c$ hold since there are no rules for either b or c

2. $\neg a$ and a hold from 1 and rules (i) and (ii)

3. $\sim a$ and $\sim\neg a$ hold from 2 and inference rule (1) (cf. page 6)

4. d and e hold from 2 and rules (iii) and (iv)

5. $\sim d$ and $\sim e$ hold from 3 and rules (iii) and (iv), as they are the only rules for d and e

6. $\sim\neg d$ and $\sim\neg e$ hold from 4 and inference rule (1) (cf. page 6).

The whole set of literals is then $\{\sim b, \sim\neg b\ \sim c, \sim\neg\neg a, a, \sim a, \sim\neg a, d, e, \sim d, \sim e, \sim\neg d, \sim\neg e\}$. □

N. B. We extend the language with the special symbol $\bot$. For every pair of classical literals $\{L, \neg L\}$ in the language of P we implicitly assume a rule $\bot \leftarrow L, \neg L$[8].

Definition 2.7 (Pseudo–interpretation). A pseudo–interpretation (p–interpretation for short) is a possibly contradictory set of ground literals from the language of a program.

In the appendix, we extend the Θ operator [36] from the class of interpretations to the class of p–interpretations, in order to define the pseudo well founded model (p–model) as the set of all literals which are consequences of a (possibly contradictory) program.

Now we have to identify sets of default literals true by CWA, whose revision to undefined can remove contradiction, by withdrawing the support of the CWAs on which the contradiction rests.

We must identify how the truth of a literal depends on the truth of other literals, i.e. how a literal L leans on a set of literals. These sets are called the *dependency sets* of L, $DS(L)$. Intuitively[9] a classical literal A will be true if there is some rule in P with A as head such that all literals in its body are also true. A default literal $\sim A$ will be true if either it has no rules, or all rules in P with head A have a false body, or the classical literal $\neg A$ is true.

[8] This is not strictly necessary but simplifies the exposition. Furthermore, without loss of generality, we only consider rules $\bot \leftarrow L, \neg L$ for which rules for both L and $\neg L$ exist in P. We also use the notation $\bot_L$ to denote the head of rule $\bot \leftarrow L, \neg L$.

[9] See definition A.7 on page 46 for the formal definition.

Example 6. (**cont.**)$P = \{a \leftarrow\sim b\ ; \neg a \leftarrow\sim c\ ; d \leftarrow a\ ; e \leftarrow \neg a\}$. In this case we have the following dependency sets:

$$
\begin{array}{llll}
DS(\sim b) & = \{\sim b\} & DS_1(\sim a) & = \{\sim a, b\} \\
DS(\sim c) & = \{\sim c\} & DS_2(\sim a) & = \{\sim a, \neg a, \sim c\} \\
DS(a) & = \{a, \sim b\} & & \\
DS(\neg a) & = \{\neg a, \sim c\} & & \\
DS_1(\sim d) & = \{\sim d\} \cup DS_1(\sim a) & = \{\sim d, \sim a, b\} & \\
DS_2(\sim d) & = \{\sim d\} \cup DS_2(\sim a) & = \{\sim d, \sim a, \neg a, \sim c\} & \\
DS(\bot_a) & = \{\bot_a, a, \neg a, \sim b, \sim c\} & &
\end{array}
$$

□

However it is not enough to establish how a literal depends on its dependency sets. We are interested in those dependency sets of a literal L which belong to the p–model, which means that L also belongs to the p–model. These sets are called the supports of L, i.e. :

Definition 2.8 (Support of a literal). A support $SS_M(L)$ w.r.t. a model M is a non–empty dependency set $DS(L)$ such that $DS(L) \subseteq M$. If there exists a $SS_M(L)$ we say that L is supported in M.

For simplicity, a support w.r.t. M_P of P can be represented by $SS(L)$. The notion of support can be extended to sets of literals:

Definition 2.9 (Support of a set of literals). A support $SS_M(\{L_1, \ldots, L_n\})$ w.r.t. a model M is:

$$SS_{M\,k}(\{L_1, \ldots, L_n\}) = \bigcup_i SS_{M\,j(i)}(L_i)$$

For each combination k of $j(i)$ there exists one support of the set of literals.

With the notion of support we are able to identify which literals support a contradiction, i.e. the literal $\bot$. In order to remove a contradiction we must change the truth value of at least one literal from each support set of $\bot$. One issue is for which literals we allow initiating change of their truth values; another is how to specify a notion of minimal change.

As mentioned before we only wish to initiate revision on default literals true by CWA, in a manner made precise later. To identify such revising literals we first define:

Definition 2.10 (Default supported). A default literal $\sim A$ is default supported w.r.t. M if **all** supports $SS_M(\sim A)$ have only default literals.

Example 7. Let $P = \{\neg a;\ a \leftarrow \sim b;\ b \leftarrow c;\ c \leftarrow d\}$. The only support of contradiction is $\{\neg a, a, \sim b, \sim c, \sim d\}$, and default supported literals are $\sim b$, $\sim c$, and $\sim d$. Here we are not interested in revising the contradiction by undefining $\sim b$ or $\sim c$ because they depend on $\sim d$. The reason is that we are attempting to remove only those contradictions based on CWAs. Now, the CWA of a literal that is supported by another depends on the CWA of the latter. □

Definition 2.11 (Self supported set). A set of default literals S is self supported w.r.t. a model M iff *there exists a* $SS_M(S) = S$.

The set of **revisable literals** induced by P is the collection R_P of all literals which belong to some self–supported set w.r.t. M_P (cf. definition A.8 in the appendix).

It is worthwhile to note (cf. proposition 4 in appendix) that if there are no rules for L nor for $\neg L$ then $\sim L$ is a revisable literal.

Example 7. (**cont.**) In this example the self supported sets w.r.t. M_P are $\{\sim b, \sim c, \sim d\}$, $\{\sim c, \sim d\}$, and $\{\sim d\}$. Thus the only revising literal is $\sim d$. □

Note how the requirement of minimality ensures that only CWA literals not depending on other CWAs are revising.

An atom can also be false by CWA if in a positive *"loop"*. Such cases are also accounted for:

Example 8. Let $P_1 = \{\neg a;\ a \leftarrow \sim b;\ b \leftarrow b, c\}$ and $P_2 = \{\neg a;\ a \leftarrow \sim b;\ b \leftarrow b;\ b \leftarrow c\}$.

For P_1 the self supported sets are: $\{\sim b, \sim c\}$, $\{\sim b\}$, and $\{\sim c\}$. Thus $\sim b$ and $\sim c$ are revisable. For P_2 the only minimal self supported set is $\{\sim c\}$ thus only $\sim c$ is revisable. Note that the only support set of $\sim b$ is $\{\sim b, \sim c\}$.

In P_2 it is clear that $\sim b$ depends on $\sim c$. So $\sim b$ is not revisable. In P_1 the truth of $\sim b$ can support on itself. Thus $\sim b$ is also revisable. □

Given the revisable literals we must find those on which the contradiction rests. This is done simply by finding the supports of $\bot$ where the revisable literals occur only as leaves (these constitute the $\bot$–**assumption sets**) – cf. definition A.10 in the appendix. In the examples shown later assumption sets are always sets of default literals $\sim A$ such that no rule for A exists in P (a very common simplifying case).

Definition 2.12. A program P is *revisable* iff no assumption set for $\bot$, $AS(\bot)$, is empty.

This definition entails a program P is not revisable if $\bot$ has some support without default supported literals, as shown in the next example:

Example 9. Consider $P = \{\neg a;\ a \leftarrow \sim b\ ; b \leftarrow \sim c\ ; c\}$ with $M_P = \{\bot, a, \sim a, \neg a, \sim b, c\}$. The only support of $\bot$ is $SS(\bot) = \{\bot, a, \sim a, \neg a, \sim b, c\}$. Since there are in it no default supported literals $R_P = \{\}$. Thus $AS(\bot) = \{\}$ and the program is not revisable. □

Definition 2.13 (Removal set). A Removal Set (RS) of a literal L of program P is a set of literals formed by the union of one **non–empty** subset from each $AS_P(L)$.

Note that, although the program may induce revising literals, this is not enough for a program to be revisable, as shown by the following example.

Example 10. Consider the program $P = \{\neg a; a \leftarrow \sim d; \neg d \leftarrow \sim e\}$. We have $M_P = \{\sim e, \sim\neg e, \neg d, \sim d, a, \neg a, \sim\neg a, \sim a\}$ which is contradictory. The only revising literal is $\sim e$ ($\sim d$ is not default supported since $\neg d \in SS(\sim d)$) and thus $AS(\sim d) = \{\}$. We have then $AS(\bot) = \{\}$ and there is no possible revision, i.e. the program is non–revisable. □

In order to make minimal changes that preserve indissociability of literals[10] we define:

Definition 2.14 (Minimal contradiction removal sets). Let R be a minimal removal set of $\bot$. A Minimal Contradiction Removal Set of program P is the smallest set MCRS such that $R \subseteq MCRS$, where MCRS is R plus its indissociable literals.

Definition 2.15 (Contradiction removal sets). A Contradiction Removal Set (CRS) of a program P is either a MCRS or the union of MCRSs.

Example 6. (**cont.**)Consider $P = \{a \leftarrow \sim b\ ; \neg a \leftarrow \sim c\ ; d \leftarrow a\ ; e \leftarrow \neg a\}$. Since $\sim b$ and $\sim c$ are both revisable literals, we have $AS(\bot) = \{\sim b, \sim c\}$. The contradiction removal sets are:

$$\begin{array}{lll} CRS_1 = & RS_1(\bot) = & \{\sim b\} \\ CRS_2 = & RS_2(\bot) = & \{\sim c\} \\ CRS_3 = & RS_3(\bot) = & \{\sim b, \sim c\} \end{array}$$

□

2.3.1. Contradiction Free Programs

In this section we show that for each contradiction removal set there is a non–contradictory program obtained from the original one by a simple update, and based on these programs we define the CRSX semantics.

[10] A class of literals identified by the CRSX theory which do not appear in any examples in this paper. Informally, *indissociable literals* are those that depend only on each other (as in positive loops), so that their truth value must always be the same. In [29] it is shown that it is impossible to change the truth value of one without changing the truth value of another. See the appendix for the formal definition as well as examples.

Definition 2.16 (CWA inhibition rule). The CWA inhibition rule for an atom A is $A \leftarrow\sim A$.

Any program P containing a CWA inhibition rule for atom A has no models containing $\sim A$[11].

Definition 2.17 (Contradiction free program). For each contradiction removal set CRS_i of a program P we engender the contradiction free program:

$$P_{CRS_i} =_{def} P \cup \{A \leftarrow\sim A | \sim A \in CRS_i\} \tag{2}$$

Proposition 1. Every contradiction free program is non–contradictory, i.e. it has WFSX semantics.

Example 11. Consider the program P :

$$\begin{array}{rlrl} a \leftarrow & \sim b & \neg c \leftarrow & \sim d \\ b \leftarrow & \sim a, \sim c & c \leftarrow & \sim e \end{array}$$

The well founded model is $M_P = \{\bot_c, \sim d, \sim\neg d, \sim e, \sim\neg e, \neg c, c, \sim\neg c, \sim c, \sim b, \sim\neg b, a, \sim\neg a\}$. The contradiction removal sets are:

$$CRS_1 = \{\sim d\} \quad CRS_2 = \{\sim e\} \quad CRS_3 = \{\sim d, \sim e\}$$

with CRS_1 and CRS_2 being minimal w.r.t. set inclusion.

- $P_{CRS_1} = P \cup \{d \leftarrow\sim d\}$ and $M_{P_{CRS_1}} = \{\sim e, \sim\neg e, c, \sim\neg c, \sim b, \sim\neg b, a, \sim\neg a, \sim\neg d\}$
- $P_{CRS_2} = P \cup \{e \leftarrow\sim e\}$ and $M_{P_{CRS_2}} = \{\sim d, \neg c, \sim c, \sim\neg a, \sim\neg b, \sim\neg d, \sim\neg e\}$
- $P_{CRS_3} = P \cup \{e \leftarrow\sim e, d \leftarrow\sim d\}$ and $M_{P_{CRS_3}} = \{\sim\neg a, \sim\neg b, \sim\neg e, \sim\neg d\}$

□

Definition 2.18 (CRSX Semantics). Given a revisable contradictory program P let CRS_i be any contradiction removal set for P. An interpretation I is a CRSX model of P iff:

$$I = \Phi_{P_{CRS_i}}(I) \tag{3}$$

The least (w.r.t. $\subseteq$) CRSX model of P is called the CRWFM model[12].

[11] This rule can be seen as the *productive* integrity constraint $\leftarrow\sim A$. In fact, since the WF Semantics implicitly has in it the *productive* constraint $\leftarrow A, \sim A$, the inhibition rule can be seen as the minimal way of expressing by means of a program rule that $\sim A$ leads to an inconsistency, and forcing A not to be false.

[12] In [29] it is proven that this model always exists.

The contradiction removal semantics for logic programs extended with explicit negation is defined by the models satisfying equation (3), which represent the different forms of revising a contradictory program.

Equation (3) says that the revised models of a revisable contradictory program are expressed as models of revised programs (using some contradiction removal set) when WFSX semantics is used.

Example 8. (**cont.**)For program P_1 the only assumption set is $AS = \{\sim b\}$. Thus the only CRS is $\{\sim b\}$, and the only CRSX model is $\{\neg a, \sim a, \sim c\}$.

For program P_2 the only assumption set is $AS_2 = \{\sim c\}$. Thus the only CRS is $\{\sim c\}$, and the only CRSX model is $\{\neg a, \sim a\}$. □

3. Summary of our representation method

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

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

  $$B \leftarrow A, \sim ab.$$

  where $\sim ab$ is a new predicate symbol. As an example consider the rule "Normally birds fly". Its representation is:

  $$fly(X) \leftarrow bird(X), \sim ab(X).$$

  Defeasible Facts are a special case of *Defeasible Rules* where A is absent.
- **Exceptions to Defeasible Rules** *Under certain conditions COND there are exceptions to the defeasible rule* $H_1 \leftarrow B_1, \sim ab_1$.

  $$ab_1 \leftarrow COND.$$

  As an example, the representation of the exception "Penguins are exceptions to the "normally birds fly" rule (i.e. rule $f \leftarrow b, \sim abb$)" is:

  $$abb \leftarrow penguin.$$

Preference rules are a special kind of exception to defeasible rules:

- **Preference Rules** *Under conditions COND, prefer to apply the defeasible rule* $H_1 \leftarrow B_1, \sim ab_1$ *instead of the defeasible rule* $H_2 \leftarrow B_2, \sim ab_2$.

  $$ab_1 \leftarrow COND, \sim ab_2.$$

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

  $$ab_b \leftarrow penguin(X), \sim ab_penguin(X).$$

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

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

- **Hypothetical (or possibly applicable) Rules** *Rule "If A then B" may or may not apply.* Its representation is:

  $$\begin{array}{rcl} B & \leftarrow & A, hyp \\ hyp & \leftarrow & \sim\neg hyp \\ \neg hyp & \leftarrow & \sim hyp \end{array}$$

  where hyp is a new predicate symbol. As an example consider the rule "Quakers might be pacifists". Its representation is:

  $$pacifist(X) \leftarrow quaker(X), hypqp(X).$$

  $$\begin{array}{rcl} hypqp(X) & \leftarrow & \sim\neg hypqp(X). \\ \neg hypqp(X) & \leftarrow & \sim hypqp(X). \end{array}$$

4. Explicit Negative Information

In this section we show the advantage of introducing explicit negation for capturing knowledge representation in commonsense reasoning problems.

Example 12. For instance, we represent rules such as *birds fly* and *rabbits are not birds* and c is a rabbit and b is a bird as P :

$$\begin{array}{rcl} f(X) & \leftarrow & b(X) \\ \neg b(X) & \leftarrow & r(X) \\ r(c) & & \\ b(b) & & \end{array}$$

From $r(c)$ follows $\neg b(c)$, and hence $\sim b(c)$, and so $\sim f(c)$. From $b(b)$ follows $f(b)$ and $\sim\neg b(b)$.

$$M_P = \left\{ \begin{array}{cccccc} b(b), & \sim\neg b(b), & \sim r(b), & \sim\neg r(b), & f(b), & \sim\neg f(b), \\ \neg b(c), & \sim b(c), & r(c), & \sim\neg r(c), & \sim f(c), & \sim\neg f(c) \end{array} \right\}.$$

Indeed, the factorization P/M_P is (trivially):

$$\begin{array}{rlrl} f(b) & \leftarrow b(b) & f(c) & \leftarrow b(c) \\ \neg b(b) & \leftarrow r(b) & \neg b(c) & \leftarrow r(c) \\ b(b) & & r(c) & \end{array}$$

and we have:

$$least(P/M_P) = \left\{ \begin{array}{cccccc} b(b), & \sim\neg b(b), & \sim r(b), & \sim\neg r(b), & f(b), & \sim\neg f(b), \\ \neg b(c), & \sim b(c), & r(c), & \sim\neg r(c), & \sim f(c), & \sim\neg f(c) \end{array} \right\} = M_P.$$

□

5. Defeasible Reasoning

In this section we show how to represent defeasible reasoning with logic programs extended with explicit negation. We want to express defeasible reasoning and give a meaning to sets of rules, (some of them being defeasible) when contradiction arises from the application of the defeasible rules. In this case we suggest how to explicitly represent exceptions and preference rules. We do not intend to address the problem of automatic generation of exception rules or preference rules[13] in order to restore consistency, but only to show how exceptions and preferences may be represented in the language. For instance, we want to represent defeasible rules such as *birds normally fly* and *penguins normally don't fly*. Given a penguin, which is a bird, we adopt the skeptical point of view and none of the conflicting rules applies. Later on we show how to express preference for one rule over another in case they conflict and both are applicable. Consider for the moment a simpler version of this problem:

Example 13. Consider the statements:

(*i*) *Normally birds fly.* (*ii*) *Penguins don't fly.*
(*iii*) *Penguins are birds.* (*iv*) `a` *is a penguin.*

[13] See [18] where an implicit preference for negative information over positive information is introduced in the semantics of logic program

represented by the program P (with obvious abbreviations, where ab stands for abnormal):

$$\begin{array}{rcll} f(X) & \leftarrow & b(X), {\sim}ab(X) & \text{(i)} \\ \neg f(X) & \leftarrow & p(X) & \text{(ii)} \\ b(X) & \leftarrow & p(X) & \text{(iii)} \\ p(a) & & & \text{(iv)} \end{array}$$

Since there are no rules for $ab(a)$, ${\sim}ab(a)$ holds and $f(a)$ follows. On the other hand we have $p(a)$ and $\neg f(a)$ follows from rule (ii). Thus M_P is contradictory. In this case we argue that the first rule gives rise to a contradiction depending on a CWA on $ab(a)$ and so must not conclude $f(a)$. The intended meaning requires $\neg f(a)$ and ${\sim}f(a)$. We say that in this case a revision occurs in the CWA of predicate instance $ab(a)$, which must turn to be undefined. ${\sim}f(a)$ follows from $\neg f(a)$ in the semantics.

In this case CRSX identifies one contradiction removal set $CRS = \{{\sim}ab(a)\}$. The corresponding contradiction free program is $P \cup \{ab(a) \leftarrow {\sim}ab(a)\}$, and the corresponding CRWFM is $\{p(a), {\sim}\neg p(a), b(a), {\sim}\neg b(a), \neg f(a), {\sim}f(a), {\sim}\neg ab(a)\}$. □

In the example above the revision process is simple and the information to be revised is clearly the CWA about the abnormality predicate, and something can be said about a flying. However this is not always the case, as shown in the following example:

Example 14. Consider the following statements: (*i*) *Normally birds fly.* (*ii*) *Normally penguins don't fly.* (*iii*) *Penguins are birds. There is a penguin* a*, a bird* b*, and a rabbit* c *which does not fly.* The program P corresponding to this description is:

$$\begin{array}{rcll} f(X) & \leftarrow & b(X), {\sim}ab_1(X) & \text{(i)} \\ \neg f(X) & \leftarrow & p(X), {\sim}ab_2(X) & \text{(ii)} \\ b(X) & \leftarrow & p(X) & \text{(iii)} \end{array} \qquad \begin{array}{r} p(a) \\ b(b) \\ r(c) \\ \neg f(c) \end{array}$$

Remark 5.1. *In program* P *above the facts and rule* (*iii*) *play the rôle of non–defeasible information, and should hold whichever the world view one may choose for the interpretation of* P *together with those facts.*

- About the bird b everything is well defined and we have:

$$\left\{ \begin{array}{cccccc} {\sim}p(b), & b(b), & {\sim}r(b), & {\sim}ab_1(b), & {\sim}ab_2(b), & f(b) \\ {\sim}\neg p(b), & {\sim}\neg b(b), & {\sim}\neg r(b), & {\sim}\neg ab_1(b), & {\sim}\neg ab_2(b), & {\sim}\neg f(b) \end{array} \right\}$$

  which says that bird b flies, $f(b)$, and it can't be shown it is a penguin, ${\sim}p(b)$. This is the intuitive result, since we may believe that b flies (because it is a bird) and it is not known to be a penguin, and so rules (i) and (ii) are non–contradictory w.r.t. bird b.

- About the penguin a use of rules (i) and (ii) provoke a contradiction in M_P: by rule (i) we have $f(a)$ and by rule (ii) we have $\neg f(a)$. Thus nothing can be said for sure about a flying or not, and the only non–ambiguous conclusions we may infer are:

$$\left\{ \begin{array}{ccccc} p(a), & b(a), & \sim r(a), & & \\ \sim\neg p(a), & \sim\neg b(a), & \sim\neg r(a), & \sim\neg ab_1(a), & \sim\neg ab_2(a) \end{array} \right\}$$

  Note that we are being skeptical w.r.t. $ab_1(a)$ and $ab_2(a)$ whose negation by CWA would rise a contradiction.

- About c rules (i) and (ii) are non-contradiction producing since $\sim p(c)$ and $\sim b(c)$ both hold, and we have:

$$\left\{ \begin{array}{cccccc} \sim p(c), & \sim b(c), & r(c), & \sim ab_1(c), & \sim ab_2(c), & \neg f(c) \\ \sim\neg p(c), & \sim\neg b(c), & \sim\neg r(c), & \sim\neg ab_1(c), & \sim\neg ab_2(c), & \sim f(c) \end{array} \right\}$$

The view of the world given by the least p–model M_P of P using WFSX is[14]:

$$\left\{ \begin{array}{ccccccc} p(a), & b(a), & \sim r(a), & \sim ab_1(a), & \sim ab_2(a), & f(a), & \neg f(a), \\ \sim\neg p(a), & \sim\neg b(a), & \sim\neg r(a), & \sim\neg ab_1(a), & \sim\neg ab_2(a), & \sim\neg f(a), & \sim f(a), \\ \sim p(b), & b(b), & \sim r(b), & \sim ab_1(b), & \sim ab_2(b), & f(b), & \\ \sim\neg p(b), & \sim\neg b(b), & \sim\neg r(b), & \sim\neg ab_1(b), & \sim\neg ab_2(b), & \sim\neg f(b), & \\ \sim p(c), & \sim b(c), & r(c), & \sim ab_1(c), & \sim ab_2(c), & & \neg f(c), \\ \sim\neg p(c), & \sim\neg b(c), & \sim\neg r(c), & \sim\neg ab_1(c), & \sim\neg ab_2(c), & & \sim f(c) \end{array} \right\}$$

A contradiction arises about penguin a ($f(a)$ and $\neg f(a)$ both hold) because of the (closed world) assumptions on $ab_1(a)$ and $ab_2(a)$, thus suggesting that the contradiction removal set is $CRS = \{\sim ab_1(a), \sim ab_2(a)\}$.

Let us determine formally the CRS we presented above. The unique contradiction arising is due to penguin a, so it is enough to consider dependency sets concerning a. The pseudo model M_P of P contains $\{f(a), \sim\neg f(a), \neg f(a), \sim f(a), b(a), \sim\neg b(a), p(a), \sim \neg p(a), \sim ab_1(a), \neg f(a), \sim ab_2(a)\}$. The dependency sets are:

$$DS(\bot_{f(a)}) = DS(f(a)) \cup DS(\neg f(a)) = \{f(a), b(a), \sim ab_1(a), \neg f(a), p(a), \sim ab_2(a)\}$$

and

$$\begin{array}{rl} DS(\sim ab_1(a)) = & \{\sim ab_1(a)\} \\ DS(\sim ab_2(a)) = & \{\sim ab_2(a)\} \end{array}$$

[14]Note that the difference between the M_P model presented and the set of literals considered as the intuitive result in the previous remark differ precisely in the truth valuation of predicate instances $ab_1(a)$, $ab_2(a)$ and $f(a)$.

Since $DS(\bot_{f(a)}) \subseteq M_P$ we have $SS(\bot_{f(a)}) = DS(\bot_{f(a)})$. The assumption set is $AS(\bot_{f(a)}) = \{\sim ab_1(a), \sim ab_2(a)\}$ because both $\sim ab_1(a)$ and $\sim ab_2(a)$ are negatively supported. The contradiction removal sets are thus:

$$\begin{array}{ll} CRS_1 = & \{\sim ab_1(a)\} \\ CRS_2 = & \{\sim ab_2(a)\} \\ CRS_3 = & \{\sim ab_1(a), \sim ab_2(a)\} \end{array}$$

with CRS_3 corresponding to the most cautious meaning (i.e. no preferred assumptions are made about abnormalities involving a).

Let us check that $M_{P_{CRS_3}}$ is a *CRXSM* model of P. Factorization $P_{CRS_3}/M_{P_{CRS_3}}$ yields:

$$\begin{array}{rcl rcl rcl} f(b) & \leftarrow & b(b) & f(a) & \leftarrow & b(a), \mathbf{u} & f(c) & \leftarrow & b(c) \\ \neg f(b) & \leftarrow & p(b) & \neg f(a) & \leftarrow & p(a), \mathbf{u} & \neg f(c) & \leftarrow & p(c) \\ b(b) & \leftarrow & p(b) & b(a) & \leftarrow & p(a) & b(c) & \leftarrow & p(c) \\ b(b) & \leftarrow & & p(a) & \leftarrow & & r(c) & \leftarrow & \\ & & & ab_1(a) & \leftarrow & \mathbf{u} & & & \\ & & & ab_2(a) & \leftarrow & \mathbf{u} & & & \end{array}$$

and $\Phi_{P_{CRS}}(M_{P_{CRS}}) = M_{P_{CRS}}$. □

5.1. Exceptions

In general we may want to say that a given element is an exception to a normality rule. The notion of exception may be expressed in two different ways.

5.1.1. Exceptions to predicates

Example 15. We express that the rule $flies(X) \leftarrow bird(X)$ applies whenever possible but can be defeated by exceptions using the rule:

$$flies(X) \leftarrow bird(X), \sim ab(X)$$

If there is a bird b and a bird a which is known not to fly (and we don't know the reason why) we may express it by $\neg flies(a)$. In this case $\neg flies(a)$ establishes an *exception to the conclusion predicate* of the defeasible rule, and the meaning of the program[15] is:

$$\left\{ \begin{array}{lllll} bird(b), & \sim ab(b), & \sim\neg ab(b), & \sim\neg bird(b), & \sim\neg flies(b), \quad flies(b), \\ bird(a), & & \sim\neg ab(a), & \sim\neg bird(a), & \neg flies(a), \quad \sim flies(a) \end{array} \right\}$$

□

[15] This is a simplified version of example 13.

Note that nothing is said about $ab(a)$, i.e. the CWA on $ab(a)$ is avoided ($\{\sim ab(a)\}$ is the CRS) since it would give rise to a contradiction on $flies(a)$. This is the case where we know that bird a is an exception to the *normally birds fly* rule, by observation of the fact that it does not fly: $\neg flies(a)$.

5.1.2. Exceptions to rules

A different way to express that a given animal is some exception is to say that a given rule must not be applicable to the animal. To state that an element is an exception to a specific rule rather than to its conclusion predicate (more than one rule may have the same conclusion), we state that the element is abnormal w.r.t. the rule, i.e. the rule is not applicable to the element: if element a is an exception to the flying birds rule we express it as $ab(a)$.

In general we may want to express that a given X is abnormal under certain conditions. This is the case where we want to express penguins are abnormal w.r.t. the flying birds rule above, as follows:

$$ab(X) \leftarrow penguin(X) \tag{4}$$

Remark 5.2. *Rule (4) together with the non–defeasible rule* $bird(X) \leftarrow penguin(X)$ *add that* penguins are birds which are abnormal w.r.t. flying.

Similarly of dead birds; i.e.

$$ab(X) \leftarrow bird(X), dead(X)$$

adding that *dead birds are abnormal w.r.t. flying.*

Remark 5.3. *Alternatively, given* $\neg flies(X) \leftarrow dead(X)$*, the non–abnormality of dead bird* a *w.r.t. flying, i.e.* $\sim ab(a)$*, may not be consistently assumed since it leads to a contradiction regarding* $flies(a)$ *and* $\neg flies(a)$.

5.1.3. Exceptions to exceptions

In general we may extend the notion of exceptioned rules to exception rules themselves, i.e. exception rules may be defeasible. This will allow us to express an exception to the exception rule for birds to fly, and hence the possibility that an exceptional penguin may fly, or that a dead bird may fly. In this case we want to say that the exception rule is itself a defeasible rule:

$$ab(X) \leftarrow bird(X), dead(X), \sim ab_deadbird(X)$$

5.2. Preferences among rules

We may express now preference between two rules, stating that if one rule may be used, that constitutes an exception to the use of the other rule:

Example 16. Consider again the flying birds example:

$$\begin{array}{rcll} f(X) & \leftarrow & b(X), {\sim}ab_1(X) & \text{(i)} \\ \neg f(X) & \leftarrow & p(X), {\sim}ab_2(X) & \text{(ii)} \\ b(X) & \leftarrow & p(X) & \text{(iii)} \end{array}$$

In some cases we want to apply the most specific information; above, there should be (since a penguin is a specific kind of bird) an explicit preference of the non–flying penguins rule over the flying birds rule:

$$ab_1(X) \leftarrow p(X), {\sim}ab_2(X) \tag{5}$$

If we have also $penguin(a)$ and $bird(b)$ the unique model contains:

$$\left\{ \begin{array}{cccccc} p(a), & b(a), & ab_1(a), & {\sim}f(a), & & {\sim}ab_2(a), \\ {\sim}p(b), & b(b), & {\sim}ab_1(b), & f(b), & {\sim}\neg f(b), & {\sim}ab_2(b) \end{array} \right\}$$

□

Rule (5) says that if a given penguin is not abnormal w.r.t. non–flying then it must be considered abnormal w.r.t. flying. In this case we infer that b is a flying bird, and a is a penguin and also a bird, and there is no evidence (assume it is false) that it flies ${\sim}f(a)$.

6. Representation of Hierarchical Taxonomies

In this section we illustrate how to represent taxonomies with logic programs with explicit negation. In this representation we wish to express general absolute (i.e. non–defeasible) rules, defeasible rules, exceptions to defeasible rules, as well as exceptions to exceptions, explicitly making preferences among defeasible rules. As we've seen, when defeasible rules contradict each other and no preference rule is present, none of them is considered applicable in the most skeptical reading. We want to be able to express preference for one defeasible rule over another whenever they conflict. In taxonomic hierarchies we wish to express that in the presence of contradictory defeasible rules we prefer the one with most specific[16] information (e.g. for a penguin, which is a bird, we want to conclude that it doesn't fly).

[16] Nute in [25] suggests using this notion of more specific information to resolve conflicts between contradictory defeasible rules.

Example 17. The statements about the domain are:

(1) Mammals are animals.
(2) Bats are mammals.
(3) Birds are animals.
(4) Penguins are birds.
(5) Dead animals are animals.
(6) Normally animals don't fly.
(7) Normally bats fly.
(8) Normally birds fly.
(9) Normally penguins don't fly.
(10) Normally dead animals don't fly.

and the following elements:

(11) Pluto is a mammal.
(12) Tweety is a bird.
(13) Joe is a penguin.
(14) Dracula is a bat.
(15) Dracula is a dead animal.

depicted as in fig. 1, and the preferences:

(16) Dead bats do not fly though bats do.
(17) Dead birds do not fly though birds do.
(18) Dracula is an exception to the above preferences.

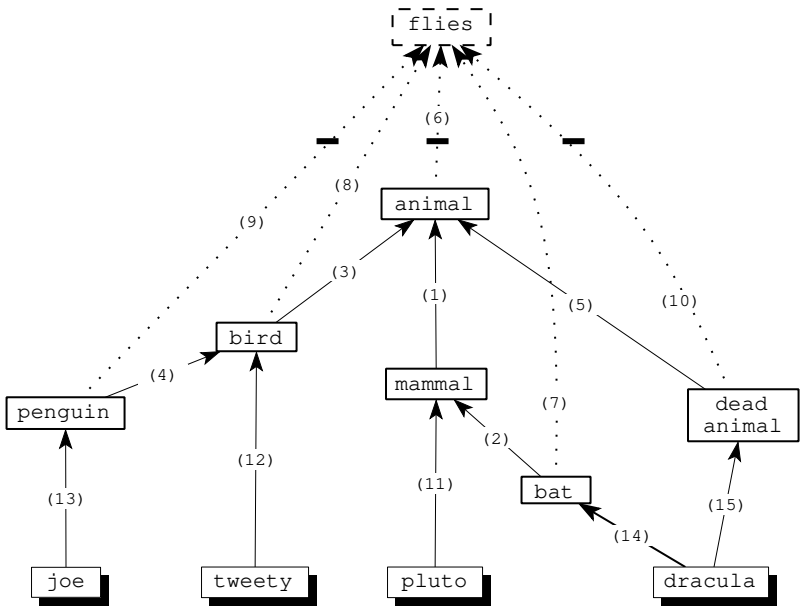

Figure 1: A more complex hierarchy

Our representation of the hierarchy is the program:

$$
\begin{aligned}
animal(X) &\leftarrow mammal(X) && (1)\\
mammal(X) &\leftarrow bat(X) && (2)\\
animal(X) &\leftarrow bird(X) && (3)\\
bird(X) &\leftarrow penguin(X) && (4)\\
animal(X) &\leftarrow dead_animal(X) && (5)\\
\neg flies(X) &\leftarrow animal(X), \sim ab_1(X) && (6)\\
flies(X) &\leftarrow bat(X), \sim ab_2(X) && (7)\\
flies(X) &\leftarrow bird(X), \sim ab_3(X) && (8)\\
\neg flies(X) &\leftarrow penguin(X), \sim ab_4(X) && (9)\\
\neg flies(X) &\leftarrow dead_animal(X), \sim ab_5(X) && (10)\\
mammal(pluto) & && (11)\\
bird(tweety) & && (12)\\
penguin(joe) & && (13)\\
bat(dracula) & && (14)\\
dead_animal(dracula) & && (15)
\end{aligned}
$$

with the implicit hierarchical preference rules (not shown in fig. 1):

$$
\begin{aligned}
ab_1(X) &\leftarrow bat(X), \sim ab_2(X)\\
ab_1(X) &\leftarrow bird(X), \sim ab_3(X)\\
ab_3(X) &\leftarrow penguin(X), \sim ab_4(X)
\end{aligned}
$$

and the explicit problem statement preferences:

$$
\begin{aligned}
ab_2(X) &\leftarrow dead_animal(X), bat(X), \sim ab_5(X) && (16)\\
ab_3(X) &\leftarrow dead_animal(X), bird(X), \sim ab_5(X) && (17)\\
ab_5(dracula) & && (18)
\end{aligned}
$$

As expected, this program has exactly one model (coinciding with the minimal WFSX model), which is non–contradictory, no choice being possible and everything being defined in the hierarchy. The model is given by the table in figure 2 where $\surd$ means that the predicate (in the row entry) is true about the element (in the column entry), e.g. $penguin(joe)$ holds in the model.

Thus pluto doesn't fly, and isn't an exception to any of the rules; tweety flies because it's a bird and an exception to the "animals don't fly" rule; joe doesn't fly because it's a penguin and an exception to the "birds fly" rule.

Note that although dracula is a dead animal, which by default don't fly (cf. rule (10)) it is also considered an exception to this very same rule. Furthermore rule (16) saying that "dead bats normally do not fly" is also exceptioned by dracula and thus the "bats fly" rule applies and dracula flies. Note that preferences rules must be present in

individ. / predicat.	joe	dracula	pluto	tweety
$dead_animal$	$\sim$, $\sim\neg$	$\surd$, $\sim\neg$	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$
bat	$\sim$, $\sim\neg$	$\surd$, $\sim\neg$	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$
$penguin$	$\surd$, $\sim\neg$	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$
$mammal$	$\sim$, $\sim\neg$	$\surd$, $\sim\neg$	$\surd$, $\sim\neg$	$\sim$, $\sim\neg$
$bird$	$\surd$, $\sim\neg$	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$	$\surd$, $\sim\neg$
$animal$	$\surd$, $\sim\neg$	$\surd$, $\sim\neg$	$\surd$, $\sim\neg$	$\surd$, $\sim\neg$
ab_4	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$
ab_2	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$
ab_3	$\surd$, $\sim\neg$	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$
ab_1	$\sim$, $\sim\neg$	$\surd$, $\sim\neg$	$\sim$, $\sim\neg$	$\surd$, $\sim\neg$
ab_5	$\sim$, $\sim\neg$	$\surd$, $\sim\neg$	$\sim$, $\sim\neg$	$\sim$, $\sim\neg$
flies	$\neg$, $\sim$	$\surd$, $\sim\neg$	$\neg$, $\sim$	$\surd$, $\sim\neg$

Figure 2: The model of the more complex hierarchy example

order to prevent contradiction to arise, thus preference rules play the rôle of removing contradictions arising in the initial specification of the problem. □

7. Hypothetical Reasoning

In this section we capture hypothetical reasoning in CRSX and interpret the results. In hierarchies everything is defined as seen, leaving no choices available (a unique model is identified as the meaning of the program). This is not the case in hypothetical reasoning situations.

7.1. The birds world

In example 13 we showed that the cautious or skeptical revision of defeasible rules gives a minimal model where no defeasible rule is used. There are however two other (non–minimal) models corresponding to alternative (non–cautious or hypothetical) meanings of the program (corresponding to alternative defeasible rules being applied or, equivalently, alternative revisions) when different assumptions are made.

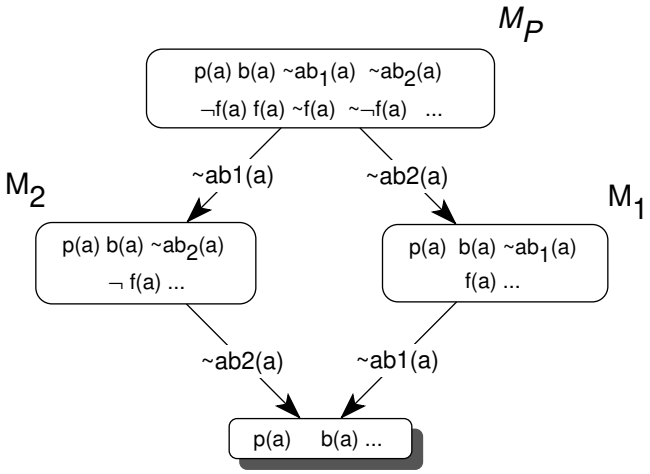

Figure 3: The birds example models

Example 18.

$$\begin{array}{rcll} f(X) & \leftarrow & b(X), \sim ab_1(X) & \text{(i)} \\ \neg f(X) & \leftarrow & p(X), \sim ab_2(X) & \text{(ii)} \\ b(X) & \leftarrow & p(X) & \\ p(a) & & & \end{array}$$

Here we may consider two alternative hypothetical worlds (note there is no preference rule present). In one of them (model M_1) we consider the hypothesis that a is not an abnormal bird, $\sim ab_1(a)$, and so it flies, $f(a)$. In this case we must also assume that $\sim ab_2(a)$ does not hold. Another alternative (model M_2) suggests that a is not an abnormal penguin, $\sim ab_2(a)$, and thus it does not fly, $\neg f(a)$. Per force, $\sim ab_1(a)$ does not hold. A third model M_3 accounts for the case where no assumption is made.

The model structure (the contradictory pseudo model M_P included) is shown in figure 3, where the shadowed model corresponds to the most skeptical view and where labels in directed edges show the revision being made from one model to the other.

Note that since M_1, M_2 and the most skeptical model M_3 are known to be non–contradictory we may use the Φ operator of WFSX. Let us check $M_1 = \{\sim\neg ab_1(a), \sim \neg b(a), \sim \neg ab_2(a), \sim \neg f(a), \sim \neg p(a), \sim ab_1(a), b(a), f(a), p(a)\}$ is a model, i.e. consider

P_{CRS_1} with $CRS_1 = \{\sim ab_2(a)\}$. P_{CRS_1}/M_1 is:

$$\begin{array}{rcll} f(a) & \leftarrow & b(a) & \text{(i)} \\ \neg f(a) & \leftarrow & p(a), \mathbf{u} & \text{(ii)} \\ b(a) & \leftarrow & p(a) & \\ p(a) & & & \\ ab_2(a) & \leftarrow & \mathbf{u} & \end{array}$$

$least(P_{CRS_1}/M_1) = \{\sim ab_1(a), b(a), f(a), p(a)\}$ and $\Phi_{P_{CRS_1}}(M_1) = \{\sim ab_1(a), b(a), \sim \neg b(a), f(a), \sim\neg f(a), p(a), \sim\neg p(a)\}$. Similarly, we have $M_2 = \{\sim ab_2(a), b(a), \neg f(a), p(a)\}$ and $CRS_2 = \{ab_1(a)\}$

$$\begin{array}{rcll} f(a) & \leftarrow & b(a), \mathbf{u} & \text{(i)} \\ \neg f(a) & \leftarrow & p(a) & \text{(ii)} \\ b(a) & \leftarrow & p(a) & \\ p(a) & & & \\ ab_1(a) & \leftarrow & \mathbf{u} & \end{array}$$

$least(P_{CRS_2}/M_2) = \{\sim ab_2(a), b(a), \neg f(a), p(a)\}$ and $\Phi_{P_{CRS_2}}(M_2) = \{\sim ab_2(a), b(a), \sim \neg b(a), \neg f(a), \sim f(a), p(a), \sim\neg p(a)\}$. Note that in both cases we have $M_3 \subseteq M_1$ and $M_3 \subseteq M_2$ where M_3 is the most skeptical view. □

Remark 7.1. *Note that every model with $\sim ab_1(a)$ is also a model with $f(a)$, i.e. $P \models\sim ab_1(a) \Rightarrow f(a)$. The same holds for the other assumption, i.e. $P \models\sim ab_2(a) \Rightarrow \neg f(a)$. Another way of interpreting these rules is by saying that if we hypothesize that, say rule (ii) has an exception in a, in the sense that $ab_2(a) \leftarrow\sim ab_2(a)$ viz. $\sim ab_2(a)$ cannot hold, then $f(a)$ holds; i.e. $P \cup \{ab_2(a) \leftarrow\sim ab_2(a)\} \models_{M_P} f(a)$.*

Compare model M_2 above with the unique model where an explicit preference was made (c.f. section 5.2 in page 22).

7.2. Hypothetical facts and rules

In some cases we want to be make a rule hypothetically applicable, in the sense that we may consider the case where the rule is used to reason with, as well as the case where the rule is not considered in force. The same is desired of some facts, i.e. we wnat to be able to explicitly represent that some unknown fact may be hypothesized true as well as false. If no hypothesis is made about the fact the information it conveys is unknown or undecided, just like the conclusion of a hypothetical rule which is not hypothesized.

7.2.1. Hypothetical 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 P_1:

$$quaker(john).\ \ pacifist(john).\ \ quaker(nixon).$$

The M_{P_1} (which is the only XM Model) is:

$$\left\{\begin{array}{cccc} & quaker(nixon) & & quaker(john) \\ \sim & pacifist(nixon) & & pacifist(john) \\ \sim\neg & quaker(nixon) & \sim\neg & quaker(john) \\ \sim\neg & pacifist(nixon) & \sim\neg & pacifist(john) \end{array}\right\}$$

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, in this or any other model (there aren't any others in fact). Now suppose we want to add:

$$Nixon\ might\ be\ a\ pacifist \tag{6}$$

In our view we wouldn't want in this case to be so strong as to affirm $pacifist(nixon)$, thereby not allowing for the possibility of Nixon not 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. Statement (6) is expressed as:

$$pacifist(nixon) \leftarrow \sim\neg pacifist(nixon) \tag{7}$$

$$\neg pacifist(nixon) \leftarrow \sim pacifist(nixon) \tag{8}$$

The first rule states that Nixon is a pacifist if there is no evidence against it. The second rule makes a symmetric statement. Let P_2 be the program P together with these rules. P_2 has a minimal model M_{P_2} (which is non–contradictory):

$$\left\{\begin{array}{cccc} & quaker(nixon) & & quaker(john) \\ & & & pacifist(john) \\ \sim\neg & quaker(nixon) & \sim\neg & quaker(john) \\ & & \sim\neg & pacifist(john) \end{array}\right\}$$

and two more XMs:

$$XSM_1 = M_{P_2} \cup \{pacifist(nixon), \sim\neg pacifist(nixon)\}$$
$$XSM_2 = M_{P_2} \cup \{\neg pacifist(nixon), \sim pacifist(nixon)\}$$

which is the result we were seeking. Statements of the form of (6) we call *unknown possible facts*, and are expressed as by (7) and (8). They can be read as a fact and its negation, each of which can be assumed only if it is consistent to do so.

7.2.2. Hypothetical rules

Consider now the well known nixon–diamond example using now hypothetical rules instead of defeasible ones.

We represent these rules as named rules (in the fashion of [35]) where the rule name may be present in one model as true, and in others as false.

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.

The corresponding logic program is:

$$
\begin{aligned}
pacifist(X) &\leftarrow quaker(X), hypqp(X)\\
hypqp(X) &\leftarrow \sim\neg hypqp(X)\\
hawk(X) &\leftarrow republican(X), hyprh(X)\\
hyprh(X) &\leftarrow \sim\neg hyprh(X)\\
\neg hawk(X) &\leftarrow pacifist(X)\\
\neg pacifist(X) &\leftarrow hawk(X)\\
quaker(nixon)&\\
republican(nixon)&\\
quaker(another_quaker)&\\
republican(another_republican)&
\end{aligned}
$$

where the following rules are also added making, each normality instance rule about Nixon hypothetical rather than defeasible (c.f. the representation of defeasible rules in section 5):

$$
\begin{array}{llll}
hypqp(nixon) & \leftarrow\sim\neg hypqp(nixon) & hyprh(nixon) & \leftarrow\sim\neg hyprh(nixon)\\
\neg hypqp(nixon) & \leftarrow\sim hypqp(nixon) & \neg hyprh(nixon) & \leftarrow\sim hyprh(nixon)
\end{array}
$$

which is represented as in fig. 4. The whole set of models is represented in fig. 5. where the models (with obvious abbreviations) are:

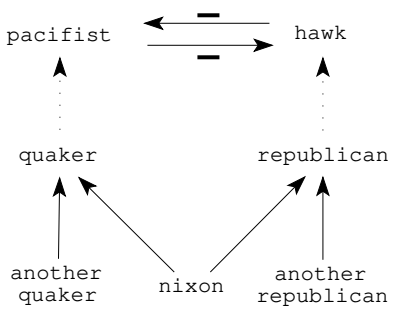

Figure 4: The Nixon diamond

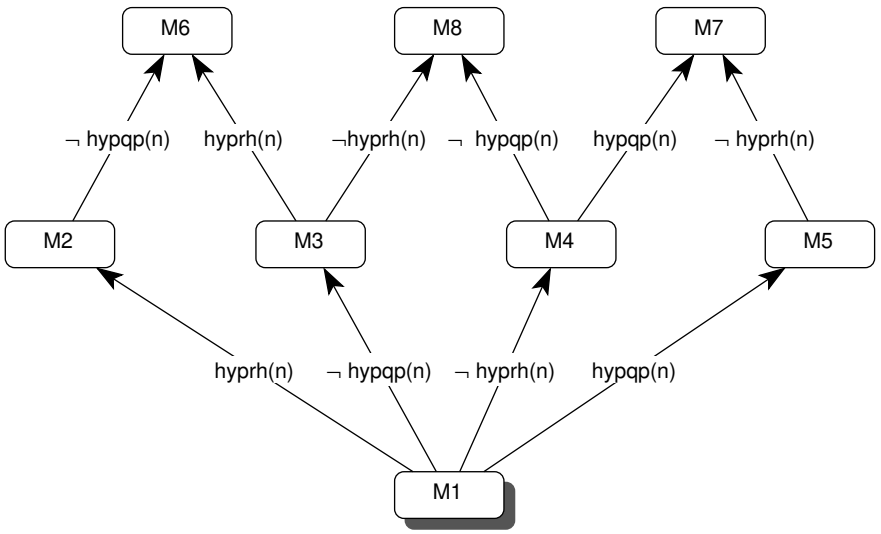

Figure 5: Models of the nixon–diamond problem using hypothetical rules

$$
\begin{array}{rrrrr}
M_1 = \{ & qua(n), & rep(n), & \sim\neg qua(n), & \sim\neg rep(n), \\
& qua(a_qua), & \sim\neg qua(a_qua), & \sim rep(a_qua), & \sim\neg rep(a_qua), \\
& hypqp(a_qua), & \sim\neg hypqp(a_qua), & pac(a_qua), & \sim\neg pac(a_qua), \\
& hyprh(a_qua), & \sim\neg hyprh(a_qua), & \sim\neg pac(a_qua), & \sim hawk(a_qua), \\
& rep(a_rep), & \sim\neg rep(a_rep), & \sim qua(a_rep), & \sim\neg qua(a_rep), \\
& hyprp(a_rep), & \sim\neg hyprp(a_rep), & rep(a_rep), & \sim\neg rep(a_rep), \\
& hypqp(a_rep), & \sim\neg hypqp(a_rep), & \sim pac(a_rep), & \sim\neg hawk(a_rep)\ \}
\end{array}
$$

$$
\begin{array}{rl}
M_2 = & M_1 \cup \{hyprh(n), \sim\neg hyprh(n), hawk(n), \sim\neg hawk(n), \neg pac(n), \sim pac(n)\} \\
M_3 = & M_1 \cup \{\neg hypqp(n), \sim hypqp(n), \sim pac(n), \sim\neg hawk(n)\} \\
M_4 = & M_1 \cup \{\neg hyprh(n), \sim hyprh(n), \sim hawk(n), \sim\neg pac(n)\} \\
M_5 = & M_1 \cup \{hypqp(n), \sim\neg hypqp(n), pac(n), \sim\neg pac(n), \neg hawk(n), \sim hawk(n)\} \\
M_6 = & M_2 \cup \{\neg hypqp(n), \sim hypqp(n), \sim pac(n), \sim\neg hawk(n)\} \\
M_7 = & M_4 \cup \{hypqp(n), \sim\neg hypqp(n), pac(n), \sim\neg pac(n), \sim\neg hawk(n)\} \\
M_8 = & M_3 \cup \{\neg hyprh(n), \sim hyprh(n), \sim hawk(n), \sim\neg pac(n)\}
\end{array}
$$

M_1 being the most skeptical one. Edge labels represent the hypothesis being made when going from one model to another.

Note that possible rules are different from defeasible rules. Defeasible rules are applied "whenever possible" unless they lead to a contradiction. Possible rules provide equally plausible alternative extensions. In the most cautious model no hypotheses are made about the applicability of normality rules. A model exists considering the applicability of the *republicans are hawks* normality rule as well as another model (M_3) considering the non–use of it. Note that M_1 and M_3 differ precisely in the way the rule is interpreted. In some sense M_3 is a model where the normality rule is not considered at all while in M_1 although the rule is considered, it is not applied since there are other equally applicable rules which together with it would give rise to an inconsistency.

Remark 7.2. *Note that with this form of representation we might as well add $abqp$ or $\neg abqp$, and thus the treatment of explicit negative information becomes similar to that of positive information. In this case we may now hypothesize about the applicability and non–applicability of each normality rule. However, the most skeptical model (where no hypotheses are made) is still identical to the one where normality rules were interpreted as defeasible rules, the difference being that in this case revision is enforced since the M_P model is non–contradictory.*

In this form of representation of the nixon–diamond problem there is no need for revision since all models are non–contradictory.

8. Application to Reasoning about Actions

We now apply the programing methodology described above to some reasoning about action problems and show that it gives correct results. The situation calculus notation [23] is used, where predicate $holds(P, S)$ expresses that property or fluent P holds in situation S; predicate $normal(P, E, S)$ expresses that in situation S, event or action E does not normally affect the truth value of fluent P; the term $result(E, S)$ names the situation resulting from the occurrence of event E in situation S.

8.1. The Yale Shooting Problem

This problem, supplied in [10], will be represented in a form nearer to the one suggested in [18].

Example 19. 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.

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

- After an event things normally remain as they were (frame axioms), i.e.:
  - properties which hold before will normally still hold after the event:

$$holds(P, result(E, S)) \leftarrow holds(P, S), \sim ab(P, E, S) \ (pp)^{17}$$

  - properties which do not hold before the event will normally not hold afterwards as well:

$$\neg holds(P, result(E, S)) \leftarrow \neg holds(P, S), \sim ab(\neg P, E, S) \ (np)^{18}$$

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:

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

[17] pp stands for positive persistence.
[18] np stands for negative persistence.

With this formulation the M_P model is the only XSM model. The subset of its elements that match with at least one of the queries is[19]:

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

which means that in situation s3 the gun is loaded and the person is not alive. This result coincides with the one obtained in [16] for *holds*. □

8.2. Multiple Extensions

Example 20. To get the result given by circumscription [22] and default logic [40], 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 (or might not) be loaded.

This clearly means an unknown but hypothetical application of (pp). So the rules to add are:

$$\begin{array}{rcl} ab(loaded, wait, S) & \leftarrow\sim & \neg ab(loaded, wait, S) \\ \neg ab(loaded, wait, S) & \leftarrow & \sim ab(loaded, wait, S) \end{array}$$

Now the M_P model contains $\sim\neg holds(loaded, s3)$. This means that in M_P 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 M_P. The rules above state that it is equally possible for *load* to be abnormal with respect to the *wait* event, as well as to be non–abnormal. We have two XMs corresponding to the two extensions. One extension contains:

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

and the other contains:

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

□

8.3. Other Reasoning About Action Problems

In this section we represent problems D2 and D6 of [19] which are classified as "Reasoning about Action - Temporal Projection" and "Reasoning about Action - Temporal Explanations with Actions of Unknown Kinds", respectively.

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

Example 21. The assumption of problem D2 and its representation are as follows:

- After an action is performed things normally remain as they were:

$$\begin{array}{rl} holds(P, result(E, S)) & \leftarrow holds(P, S), \sim ab(P, E, S)\ (pp) \\ \neg holds(P, result(E, S)) & \leftarrow \neg holds(P, S), \sim ab(\neg P, E, S)\ (np) \end{array}$$

- When the robot grasps a block, the block will normally be in the hand:

$$holds(hand(B), result(grasp(B), S)) \leftarrow \sim ab(hand(B), grasp(B), S)$$

- When the robot moves a block onto the table, the block will normally be on the table:

$$holds(table(B), result(move(B), S)) \leftarrow holds(hand(B), S), \sim ab(table(B), move(B), S)$$

- Initially block A is not in the hand and not on the table.

$$\neg holds(table(a), s0) \qquad \neg holds(hand(a), s0)$$

The conclusion "After the robot grasps block a, waits, and then moves it onto the table, the block will be on the table" can be represented by:

$$\leftarrow holds(table(a), result(move(a), result(wait, result(grasp(a), s0))))$$

and belongs to the M_P model of the program. □

Example 22. The assumptions of problem D6 are those of D2 plus "After the robot performed two actions, a was on the table". The conclusion is "The first action was grasping a, and the second was moving it onto the table". We reach this conclusion by verifying that

$$\leftarrow holds(table(a), result(move(a), result(grasp(a), s0)))$$

is the only goal of the form

$$\leftarrow holds(table(a), result(Action2, result(Action1, s0)))$$

which is in M_P. □

9. Abduction within CRSX

We will begin by giving a definition of abduction within CRSX semantics very similar to the ones given for classical logic in [4, 14]. Finally we present a theorem that relates the *CRXSM*s models of a modified program with abduction within the WFSX semantics of the original program.

Definition 9.1 (Abductive theory). An abductive theory is pair $\langle P, Ab\rangle$ where P is a program and Ab is a set of literals in $\mathcal{H}_P$.

Ab is the set of all literals we are prepared to assume true if that contributes to prove some goal. We call these literals the abducible literals.

Definition 9.2 (Abductive solution). For a given abductive theory $\langle P, Ab\rangle$, a subset S of Ab whose elements are not in the CRWFM model of P is an abductive solution for literal L iff the CRWFM model of $P \cup S$ exists and contains L.

Example 23. Consider the abductive theory $\langle\{f \leftarrow b, {\sim}abb \ ; \ \neg f \leftarrow p, {\sim}abp \ ; \ b \leftarrow p \ ; \ abb \leftarrow p, {\sim}abp\}, \{b, p\}\rangle$.

The CRWFM model of $P \cup \{p\}$ contains:

$$\{p, {\sim}\neg p, b, \neg f, {\sim}f, {\sim}abp, abb, {\sim}\neg b\}$$

and p is an abductive solution for $\neg f$. □

Now that we have defined abduction within CRSX semantics, we define a modification of a given program based on the abducible literals, and then establish results between the *CRXSM*s of the modified program (abducing program) and the abductive solutions in the original one.

Definition 9.3 (Abducing program). An abducing program for an abductive theory $\langle P, Ab\rangle$ is a program obtained by adding to P, for all literals L in Ab, two rules of the form: $L \leftarrow {\sim}\neg L$ and $\neg L \leftarrow {\sim}L$.

Example 23. (**cont.**)

$$\begin{array}{llll} f \leftarrow & b, {\sim}abb & \neg f \leftarrow & p, {\sim}abp \\ b \leftarrow & p & abb \leftarrow & p, {\sim}abp \\ p \leftarrow & {\sim}\neg p & \neg p \leftarrow & {\sim}p \end{array}$$

is the abducing program for the theory

$$\langle\{f \leftarrow b, {\sim}abb; \neg f \leftarrow p, {\sim}abp; b \leftarrow p; abb \leftarrow p, {\sim}abp\} \ , \ \{p\}\rangle$$

□

Notice how the abducing program is based on adding rules to make each abducible an hypothetical fact (cf. section 7.2.1), which can be used for explanation if necessary, but otherwise has no predictive value since it remains undecided in the least model.

Theorem 9.1 (Abductive solution). S is an abductive solution for L in an abductive theory $\langle P, Ab\rangle$ iff all *CRXSM* models which contain S of the abducing program for the abductive theory $\langle P, S\rangle$ also contain L and at least one exists.

This provides a technique for performing abductive inference, namely by considering the *CRXSM* models where some desired conclusion holds.

Example 23. (**cont.**) The abducing program above has the following *CRXSM*s:

$$CRXSM_1 = \{\}$$
$$CRXSM_2 = \{p, \sim\neg p, b, \sim\neg b, \neg f, \sim f, \sim abp, \sim\neg abp, abb, \sim\neg abb\}$$
$$CRXSM_3 = \{\neg p, \sim p, \sim b, \sim abb, \sim abp, \sim\neg abb, \sim\neg abp, \sim f, \sim\neg f\}$$

And the model containing p also contains $\neg f$. □

10. Related work

In this section we compare our approach with other logic programming proposals. Since the underlying semantics is different from that of other approaches[20] we focus the discussion from the point of view of dealing with inconsistency and representing defeasible reasoning. [13] is a survey of abductive reasoning in logic programming.

When making assumptions (introduced by some non–monotonic reasoning formalism), inconsistency may arise. Some approaches have been proposed recently to deal with inconsistency. One common goal to those approaches is the identification of *maximum consistent assumption sets*. Several techniques have been suggested to solve this goal.

The technique we follow goes along the lines presented in [44][21], which roughly starts by considering every consequence of a program even those raising contradictions. In our approach we consider then dependency information, for the special atom $\bot$ to define the set of revisable literals, and indentify sets of negative assumptions (assumption sets) which may not be held together (contradiction removal sets). In [44] the disjunctive normal form of a formula is defined and, together with dependency information a model is identified as the meaning of the program.

Dung [3] studies also the problem of restoring consistency of extended logic programs (with two forms of negation), in the well founded semantics. He applies a preferred

[20] For a comparison regarding the underlying semantics and other semantics refer to [1].

[21] Although the approach suggested here and presented as a natural deduction system contains two forms of negation the semantics are however different.

extension semantics to an abductive framework obtained from the original extended logic program. The extended logic program is first transformed to a logic program renaming every explicit negative literal $\neg L$ into a new positive literal L'. The program so obtained is transformed to a suitable abductive framework renaming NAF literals $\sim L$ by L^* and adding an integrity constraint $\neg(L \wedge L^*)$ along the lines of [4], plus a second integrity constraint of the form $\neg(L \wedge L')$ expressing that a literal and its explicit negation may not both hold together. Extended preferred extensions (in the sense of [4]) identify sets of assumptions which may be consistently added (i.e. satisfying the integrity constraints).

An alternative approach to restoring consistency starts from the set of consequences of the (inconsistent) theory and retracts literals until consistency is restored. This is the approach followed in [9, 17, 29].

Exploring the relation between truth–maintenance systems and logic programming, Giordano and Martelli[8] suggest to identify the (stable model) semantics of a logic program with integrity constraints as the semantics of a transformed logic program, without integrity constraints, using all the contrapositive variants of the original program rules. The idea of using the contrapositives is also explored by Jonker [12] but considering the well founded-semantics.

Satoh and Iwayama [41] use TMS techniques to maintain consistency while computing stable models of general logic programs with integrity constraints, and apply it to compute abduction in an abductive framework (c.f. [4, 15]).

In [18] Kowalski et al. suggest an extension (e–answer sets) based on stable model semantics, dealing with two forms of negation. In the semantics suggested therein there is an a priori explicit commitment to giving preference to negative information (*exceptions*) over positive information. In our present approach we showed that there is no need to make such a commitment at the semantic level (actually we argue that it is too strong), and that preferences can be represented instead explicitly at the language level. This stance was already adopted by the present authors previously in a different semantics [34], and independently by Inoue in [11]. Moreover, the ability to state preferences at the language level provides a more general mechanism for dealing with preferences. Dealing with preferences at the language level provides a greater modularity (in the sense that it is enough to consider local changes in the new program) when introducing new preference rules into an existing logic program.

Because of its inherent asymmetry, the "rules with exceptions" approach [18] 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 explicit 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.

Acknowledgements

We thank ESPRIT BR projects COMPULOG (no. 3012), COMPULOG 2 (no. 6810) and Junta Nacional de Investigação Científica e Tecnológica for their support.

References

[1] J. J. Alferes and L. M. Pereira. On logic program semantics with two kinds of negation. In K. Apt, editor, *International Joint Conference and Symposium in Logic Programming*, pages 574–588. MIT Press, 1992.

[2] J. Dix. Classifying semantics of logic programs. In Nerode et al. [24], pages 166–180.

[3] P. M. Dung and P. Ruamviboonsuk. Well founded reasoning with classical negation. In Nerode et al. [24], pages 120–132.

[4] K. Eshghi and R. Kowalski. Abduction compared with negation by failure. In G. Levi and M. Martelli, editors, *International Conference on Logic Programming*, pages 234–254. MIT Press, 1989.

[5] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. A. Kowalski and K. A. Bowen, editors, *5th International Conference on Logic Programming*, pages 1070–1080, Seattle, 1988. MIT Press.

[6] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In Warren and Szeredi, editors, *International Conference on Logic Programming*, pages 579–597, Israel, September 1990. MIT Press.

[7] M. L. Ginsberg, editor. *Readings in Non-monotonic Reasoning*. Morgan Kaufmann Publishers, Inc., 1987.

[8] L. Giordano and A. Martelli. Generalized stable models, truth maintenance and conflict resolution. In *7th International Conference on Logic Programming*, pages 427–441, 1990.

[9] R. Goebel, K. Furukawa, and D. L. Poole. Using definite clauses and integrity constraints as the basis for a theory formation approach to diagnostic reasoning. In E. Shapiro, editor, *International Conference on Logic Programming*, pages 211–222. Springer-Verlag, 1986.

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

[11] K. Inoue. Extended logic programs with default assumptions. In Koichi Furukawa, editor, *8th International Conference on Logic Programming*, pages 490–504. MIT Press, 1991.

[12] C. M. Jonker. Cautious backtracking and well-founded semantics in truth maintenance systems. Technical report, Utrecht University-Dep. of Compuer Science, 1991.

[13] A. C. Kakas, R. A. Kowalski, and F. Toni. Abductive logic programming. Technical report, Imperial College, October 1991.

[14] A. C. Kakas and P. Mancarella. Anomalous models and abduction. In *Proc. 2nd Int. Symp. on AI*, Monterey, Mexico, 1989.

[15] A. C. Kakas and P. Mancarella. Generalized stable models:a semantics for abduction. In *9th European Conference in AI*, pages 385–391, Stockholm, 1990.

[16] R. Kowalski. Problems and promises of computational logic. In John Lloyd, editor, *Computational Logic Symposium*, pages 1–36. Springer-Verlag, November 1990.

[17] R. Kowalski and F. Sadri. Knowledge representation without integrity constraints. Technical report, Department of Computing Imperial College, 1988.

[18] R. Kowalski and F. Sadri. Logic programs with exceptions. In Warren and Szeredi, editors, *International Conference on Logic Programming*, pages 598–613, Jerusalem, Israel, 1990. MIT Press.

[19] V. Lifschitz. Benchmarks problems for formal non-monotonic reasoning. In M. Reinfrank, J. d. Kleer, M. Ginsberg, and E. Sandewall, editors, *Non Monotonic Reasoning: 2nd International Workshop -Grassau, FRG*, volume 346, pages 202–219. Springer–Verlag, 1988.

[20] J. W. Lloyd. *Foundations of Logic Programming*. Symbolic Computation. Springer–Verlag, second, extended edition, 1984.

[21] W. Marek and M. Truszczynski. Computing intersections of autoepistemic extensions. In Nerode et al. [24], pages 21–37.

[22] J. McCarthy. Applications of circumscription to formalizing common-sense knowledge. In M. Ginsberg, editor, *Readings in Non–Monotonic Reasoning*, pages 153–166. Morgan Kaufmann Publishers, Inc., 1987. Also in AI journal v28 (1986).

[23] J. McCarthy and P. J. Hayes. *Some Philosophical Problems for the Standpoint of Artificial Intelligence*, pages 26–45. In Ginsberg [7], 1987. Also in B. Meltzer and D. Mitchie (eds) *Machine Intelligence 4*, Edinburgh Univ. Press, pp. 463–502, 1969.

[24] Anil Nerode, Wiktor Marek, and V. S. Subrahmanian, editors. *First International Workshop on Logic Programming and Non-monotonic Reasoning*. MIT Press, 1991.

[25] D. Nute. Ldr : A logic for defeasible reasoning. Technical report, Advanced Computational Center, Univ. of Georgia, 1986.

[26] L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *10th European Conference on Artificial Intelligence*, pages 102–106. John Wiley & Sons, Ltd, 1992.

[27] L. M. Pereira, J. J. Alferes, and J. N. Aparício. Contradiction Removal within Well Founded Semantics. In Nerode et al. [24], pages 105–119.

[28] L. M. Pereira, J. J. Alferes, and J. N. Aparício. The extended stable models of contradiction removal semantics. In P. Barahona, L. M. Pereira, and A. Porto, editors, *EPIA '91*, pages 105–119. Springer-Verlag, 1991.

[29] L. M. Pereira, J. J. Alferes, and J. N. Aparício. Contradiction removal semantics with explicit negation. In *Proceedings of Applied Logic at Work*, LNCS. Springer–Verlag, December 1992.

[30] L. M. Pereira, J. J. Alferes, and J. N. Aparício. Default theory for well founded semantics with explicit negation. In D. Pearce and G. Wagner, editors, *Logics for AI - JELIA '92*, LNAI 633, pages 339–356. Springer–Verlag, 1992.

[31] L. M. Pereira, J. J. Alferes, and J. N. Aparício. Well founded semantics with explicit negation and default theory. Technical report, AI Center, Uninova, 1992.

[32] L. M. Pereira, J. N. Aparício, and J. J. Alferes. Counterfactual reasoning based on revising assumptions. In Vijay Saraswat and Kazunori Ueda, editors, *International Symposium in Logic Programming*, pages 566–580. MIT Press, 1991.

[33] L. M. Pereira, J. N. Aparício, and J. J. Alferes. Hypothetical reasoning with well founded semantics. In B. Mayoh, editor, *Third Scandinavian Conference on Artificial Intelligence*. IOS Press, 1991.

[34] L. M. Pereira, J. N. Aparício, and J. J. Alferes. Non-monotonic reasoning with well founded semantics. In Koichi Furukawa, editor, *International Conference on Logic Programming*, pages 475–489. MIT Press, 1991.

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

[36] H. Przymusinska and T. C. Przymusinski. *Semantic Issues in Deductive Databases and Logic Programs*, pages 321–367. Formal Techniques in Artificial Intelligence. North Holland, 1990.

[37] T. C. Przymusinski. Well-founded semantics coincides with three-valued stable semantics. Technical report, Univ. of Texas at El Paso, 1989.

[38] T. C. Przymusinski. Extended stable semantics for normal and disjunctive programs. In Warren and Szeredi, editors, *International Conference on Logic Programming*, pages 459–477. MIT Press, 1990.

[39] T. C. Przymusinski. A semantics for disjunctive logic programs. In D. Loveland, J. Lobo, and A. Rajasekar, editors, *ILPS Workshop on Disjunctive Logic Programs*, 1991.

[40] R. Reiter. A logic for default reasoning. *Journal of Artificial Intelligence*, 13:81–132, 1980. Also in [7], pages 68–93.

[41] Ken Satoh and Noboru Iwayama. Computing abduction by using the TMS. In Koichi Furukawa, editor, *International Conference on Logic Programming*, pages 505–518. MIT Press, 1991.

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

[43] G. Wagner. A database needs two kinds of negation. In B. Thalheim, J. Demetrovics, and H-D. Gerhardt, editors, *MFDBS'91*, pages 357–371. Springer-Verlag, 1991.

[44] G. Wagner. Ex contradictione nihil sequitur. In *International Joint Conference on Artificial Intelligence*, 1991.

A. CRSX Review

Definition A.1 (The Θ^x operator). Let P be a logic program and J a p–interpretation. The operator $\Theta^x_J : \mathcal{I} \rightarrow \mathcal{I}$ on the set $\mathcal{I}$ of all 3-valued p–interpretations of P is defined as follows: If $I \in \mathcal{I}$ is a p–interpretation of P and A is a ground classical literal then $\Theta^x_J(I)$ is the p–interpretation defined by:

1. $\Theta^x_J(I)(A) = 1$ iff there is a rule $A \leftarrow L_1, \ldots, L_n$ in P such that for all $i \leq n$ either $\hat{J}(L_i) = 1$, or L_i is positive and $I(L_i) = 1$;
2. $\Theta^x_J(I)(A) = 0$ iff one of the following holds:
    1. for every rule $A \leftarrow L_1, \ldots, L_n$ in P there is an $i \leq n$, such that either $\hat{J}(L_i) = 0$, or L_i is positive and $I(L_i) = 0$;
    2. $\hat{J}(\neg A) = 1$;
3. $\Theta^x_J(I)(A) = 1/2$ otherwise.

Note that the only difference between this definition and the definition of Θ operator introduced in [36] is condition (2.2) capturing the coherence requirement, or inference rule (1). Furthermore, since it is defined over the class of p–interpretations, it allows that for a given literal L, we may have $\Theta^x_J(I)(L) = 1$ as well as $\Theta^x_J(I)(L) = 0$.

Proposition 2. For every p–interpretation J, the operator Θ^x_J is monotone and has a unique least fixed point given by $\Theta^{x\uparrow w}_J$, also denoted by $\Omega^x(J)$[22].

Definition A.2 (p–model). Given a program, a p–model is a p–interpretation I such that:

$$I = \Omega^x(I) \tag{9}$$

Remark A.1. *Note that if a p–model M is contradictory w.r.t. L then M is inconsistent w.r.t. L by virtue of inference rule (1), although the converse is not true.*

Definition A.3 (Well Founded Model). The pseudo Well Founded Model M_P of P is the F–least p–model.

The non–minimal models satisfying (9) above (pseudo) Extended Models (XMs for short). To compute the p–model M_P we define the following transfinite sequence $\{I_\alpha\}$ of fixed points:

[22] Recall [36] that the F-least interpretation used to compute the least fixed point of $\Theta^{x\uparrow w}_J$ is $\sim\mathcal{H}_P$.

$$\begin{aligned} I_0 &= \langle \emptyset, \emptyset \rangle \\ I_{\alpha+1} &= \Omega^x(I_\alpha) = \Theta^x_{I_\alpha}{}^{\uparrow w} \\ I_\delta &= \bigcup_{\alpha<\delta} I_\alpha \text{ for limit ordinal } \delta \end{aligned}$$

Equivalently, the pseudo well founded model M_P of P is the F–least fixed point of (9) and is given by $M_P = I_\lambda = \Omega^{x\uparrow\lambda}$.

Definition A.4. A program P is contradictory iff $\bot \in M_P$.

Example 6. **(cont.)** $P = \{a \leftarrow \sim b\ ;\ \neg a \leftarrow \sim c\ ;\ d \leftarrow a\ ;\ e \leftarrow \neg a\}$.

$$\begin{aligned} \Theta^{\uparrow 0}_{I_0} &= \{\sim a, \sim\neg a, \sim b, \sim\neg b, \sim c, \sim\neg c, \sim d, \sim\neg d, \sim e, \sim\neg e\} \\ \Theta^{\uparrow 3}_{I_0} = \Theta_{I_0}(\Theta^{\uparrow 2}_{I_0}) &= \{\sim b, \sim c, \sim\neg b, \sim\neg c, \sim\neg d, \sim\neg e\} = \Theta^{\uparrow w}_{I_0} = I_1 \\ \Theta^{\uparrow 3}_{I_1} = \Theta_{I_1}(\Theta^{\uparrow 2}_{I_1}) &= \{a, d, \neg a, e, \sim b, \sim\neg b, \sim c, \sim\neg c, \sim\neg d, \sim\neg e\} = \Theta^{\uparrow w}_{I_1} = I_2 \\ \Theta^{\uparrow 2}_{I_2} = \Theta_{I_2}(\Theta^{\uparrow 1}_{I_2}) &= \{a, \sim\neg a, \neg a, \sim a, d, \sim\neg d, e, \sim\neg e, \sim b, \sim\neg b, \sim c, \sim\neg c, \sim d, \sim\neg d, \sim e, \sim\neg e\} \\ &= \Theta^{\uparrow w}_{I_2} = I_3 \\ \Theta^{\uparrow 2}_{I_3} = \Theta_{I_3}(\Theta^{\uparrow 1}_{I_3}) &= \{a, \sim\neg a, \neg a, \sim a, d, \sim\neg d, e, \sim\neg e, \sim b, \sim\neg b, \sim c, \sim\neg c, \sim d, \sim\neg d, \sim e, \sim\neg e\} \\ &= \Theta^{\uparrow w}_{I_3} = I_4 = I_3 \end{aligned}$$

□

Note that the set of consequences of P is inconsistent in a, d and e, although it is contradictory only in a.

Proposition 3. If the pseudo well founded model M_P is non-contradictory then it is consistent.

This suggests that in order to get revised non–contradictory consistent models we must know where contradiction arises from and prevent it.

The least 3-valued model of a non-negative program can be defined as the least fixpoint of the following generalization of the van Emden–Kowalski least model operator Ψ for definite logic programs:

Definition A.5 (Ψ^* operator). Suppose that P is a non-negative program, I is an interpretation of P and A is a ground atom. Then $\Psi^*(I)$ is an interpretation defined as follows:

- $\Psi^*(I)(A) = 1$ iff there is a rule $A \leftarrow A_1, \ldots, A_n$ in P such that $I(A_i) = 1$ for all $i \leq n$.
- $\Psi^*(I)(A) = 0$ iff for every rule $A \leftarrow A_1, \ldots, A_n$ there is an $i \leq n$ such that $I(A_i) = 0$.

- $\Psi^*(I)(A) = 1/2$, otherwise.

Definition A.6 (Least-operator). We define $least(P)$, where P is a non-negative program, as the set of literals $T \cup \sim F$ obtained as follows:

- Let P' be the non-negative program obtained by replacing in P every negative classical literal $\neg L$ by a new atomic symbol, say L'.
- Let $T' \cup \sim F'$ be the least 3-valued model of P' (as defined in [36] say).
- $T \cup \sim F$ is obtained from $T' \cup \sim F'$ by reversing the replacements above.

Definition A.7 (Dependency set). A Dependency Set of a literal L in a program P, represented as $DS(L)$, is obtained as follows:

1. If L is a classical literal:
    1. if there are no rules for L then the only $DS(L) = \{L\}$.
    2. for each rule $L \leftarrow B_1, \ldots, B_n (n \geq 0)$ in P for L, there exists one $DS_k(L) = \{L\} \cup \bigcup_i DS_{j(i)}(B_i)$ for each different combination k of one $j(i)$ for each i.
2. For a default literal $\sim L$:
    1. if there are no rules in P for L then a $DS(\sim L) = \{\sim L\}$.
    2. if there are rules for L then choose from every rule for L a single literal. For each such choice there exist several $DS(\sim L)$; each contains $\sim L$ and one dependency set of each default complement[23] of the chosen literals.
    3. if there are rules for $\neg L$ then there are, **additionally**, dependency sets $DS(\sim L) = \{\sim L\} \cup DS_k(\neg L)$ for each k.

Definition A.8 (Revising and co–Revising literals). Given a program P with pseudo well-founded model M_P, we define co–R_P, the *co–revising literals* induced by P, as the set of literals belonging to some minimal self supported set w.r.t. M_P. We define R_P, the *revising literals*, as the set of co–revising literals L such that $\neg L \notin M_P$.

Proposition 4.

1. If there are no rules for L nor for $\neg L$ then $\sim L$ is a revising literal.
2. If there are no rules for L but there are for $\neg L$ then $\sim L$ is a co–revising literal but not a revising one.

[23] The default complement of a classical literal L is $\sim L$; that of a default literal $\sim L$ is L.

3. If there are no rules for L then $\sim L$ is a co–revising literal.

Definition A.9 (Indissociable set of literals). A set of default literals S is indissociable iff it is a minimal self supported set, and S is its only support.

Example 24. Let $P = \{\neg p;\ p \leftarrow \sim a;\ a \leftarrow b;\ b \leftarrow c;\ c \leftarrow a\}$. $\{\sim a, \sim b, \sim c\}$ is a set of indissociable literals. □

Definition A.10 (Assumption set). Let P be a program with meaning M_P and $L \in M_P$. An assumption set $AS(L)$ is defined as follows, where $\mathcal{R}_P$ is the set of the revisable literals induced by program P :

1. If L is a classical literal:
    1. if there is a fact for L then the only $AS(L) = \{\}$.
    2. for each rule $L \leftarrow B_1, \ldots, B_n (n \geq 1)$ in P for L such that $\{B_1, \ldots, B_n\} \subseteq M_P$, there exists one $AS_k(L) = \bigcup_i AS_{j(i)}(B_i)$ for each different combination k of one $j(i)$ for each i.
2. For a default literal $\sim L$:
    1. if $\sim L \in \mathcal{R}_P$ then the only $AS(\sim L) = \{\sim L\}$.
    2. if $L \in co - R_P$ then there is a $AS(\sim L) = \{\sim L\}$.
    3. if there are rules for L then choose from every rule for L a single literal whose default complement belongs to M_P. For each such choice there exist several $AS(\sim L)$; each contains one assumption set of each default complement of the chosen literals.
    4. if there are rules for $\neg L$ and $\neg L \in M_P$ then there are, **additionally**, assumption sets $AS(\sim L) = AS_k(\neg L)$ for each k.

Example 25. Let P be:

$$\begin{array}{llll} a & \leftarrow \sim b & b & \leftarrow c \\ \neg a & \leftarrow \sim c & c & \leftarrow b \end{array}$$

The only self supported sets is $S = \{\sim b, \sim c\}$. Moreover the only support of S is itself. Thus $\sim b$ and $\sim c$ are revisable, and indissociable.

As the only assumption set of $\bot$ is $\{\sim b, \sim c\}$ there are three removal sets: $R_1 = \{\sim b\}$, $R_2 = \{\sim c\}$, and $R_3 = \{\sim b, \sim c\}$. Without indissociability, one might think that for this program there exist three distinct ways of removing the contradiction. This is not the case since the XSMs of P_{R_1}, P_{R_2}, and P_{R_3} are exactly the same, i.e. they all represent the same revision of P. This is accounted for by minimal contradiction removal sets. In fact there exists only one MCRS $\{\sim b, \sim c\}$ and thus the only contradiction free program is P_{R_3}. □