REVISE: An Extended Logic Programming System for Revising Knowledge Bases

Carlos Viegas Damásio
CRIA, Uninova and DCS
U. Nova de Lisboa
2825 Monte da Caparica, Portugal

Wolfgang Nejdl
Informatik V
RWTH Aachen
D-52056 Aachen, Germany

Luís Moniz Pereira
CRIA, Uninova and DCS
U. Nova de Lisboa
2825 Monte da Caparica, Portugal

Abstract

In this paper we describe REVISE, an extended logic programming system for revising knowledge bases. REVISE is based on logic programming with explicit negation, plus a two-valued assumption revision to face contradiction , encompasses the notion of preference levels. Its reliance on logic programming allows efficient computation and declarativity, whilst its use of explicit negation, revision and preference levels enables modeling of a variety of problems including default reasoning, belief revision and model-based reasoning. It has been implemented as a Prolog–meta interpreter and tested on a spate of examples, namely the representation of diagnosis strategies in modelbased reasoning systems.

1 INTRODUCTION

While a lot of research has been done in the area of nonmonotonic reasoning during the last decade, relatively few systems have been built which actually reason nonmonotonically. This paper describes the semantics and core algorithm of REVISE, a system based on an extended logic programming framework. It is powerful enough to express a wide variety of problems including various nonmonotonic reasoning and belief revision strategies and more application oriented knowledge such as diagnostic strategies in modelbased reasoning systems ([de Kleer, 1991, Friedrich and Nejdl, 1992, Lackinger and Nejdl, 1993, Dressler and Bottcher, 1992]).

We start, in Section 2, by reviewing the well founded semantics with explicit negation and two valued contradiction removal [Pereira *et al.*, 1993b], which supplies the basic semantics for REVISE. We then introduce in Section 3 the concept of preference levels amongst sets of assumptions and discuss how it integrates into the basic semantics. Section 4 gives examples of application of REVISE for describing diagnostic strategies in modelbased reasoning systems. Section 5 describes the core algorithm of REVISE, which is an extension of Reiter's algorithm, in [Reiter, 1987], for computing diagnoses in modelbased reasoning systems, corrected in [Greiner *et al.*, 1989]. Finally, Section 6 contains comparisons with related work and conclusions.

2 REVIEW OF THE LOGIC PROGRAMMING BASIS

In this section we review WFSX, the Well Founded Semantics of logic programs with eXplicit negation and its paraconsistent version. We focus the presentation on the latter. Basically, WFSX follows from WFS [Gelder *et al.*, 1991] plus one basic "coherence" requirement relating the two negations: $\neg L$ entails $\sim L$ for any literal L. We also present its two–valued contradiction removal version [Pereira *et al.*, 1993b]. For details refer to [Pereira and Alferes, 1992, Alferes, 1993].

Given a first order language $Lang$, an extended logic program (ELP) is a set of rules and integrity 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 objective literals, and in integrity rules H is $\perp$ (contradiction). An objective literal is either an atom A or its explicit negation $\neg A$, where $\neg\neg A = A$. $\sim L$ is called a default or negative literal. Literals are either objective or default ones. The default complement of objective literal L is $\sim L$, and of default literal $\sim L$ is L. A rule stands for all its ground instances wrt $Lang$. A set of literals S is non-contradictory iff there is no $L \in S$ such that $\sim L \in S$. For every pair of objective literals $\{L, \neg L\}$ in $Lang$ we implicitly assume the integrity rule $\perp \leftarrow L, \neg L$.

In order to revise possible contradictions we need first to identify those contradictory sets implied by a program under the paraconsistent WFSX. The main idea here is to compute all consequences of the program, even those leading to contradiction, as well as those arising from contradiction. The following example provides an intuitive preview of what we mean to capture:

Example 1 *Consider program* P :

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

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 the coherence principle relating the two negations*
4. d *and* e *hold from 3 and rules (iii) and (iv)*
5. $\sim d$ *and* $\sim e$ *hold from 2 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 the coherence principle.*

The whole set of literal consequences is then:

$$\{\sim b, \sim c, \neg a, a, \sim a, \sim\neg a, d, e, \sim d, \sim e, \sim\neg d, \sim\neg e\}.$$

For the purpose of defining WFSX and its paraconsistent extension we begin by defining paraconsistent interpretation.

Definition 2.1 *A p–interpretation I is any set* $T \cup \sim F$, *such that if* $\neg L \in T$ *then* $L \in F$ *(coherence).*

The definition of WFSX (in [Pereira and Alferes, 1992]) is based on a modulo transformation and a monotonic operator. On first reading the reader may now skip to definition 2.5.

Without loss of generality, and for the sake of technical simplicity, we consider that programs are always in their canonical form, i.e. for each rule of the program and any objective literal, if L is in the body then $\sim\neg L$ also belongs to the body of that rule[1].

Definition 2.2 *Let* P *be an canonical extended logic program and let* I *be a p–interpretation. By a* $\frac{P}{I}p$ *program we mean any program obtained from* P *by first non–deterministically applying the operations until they are no longer applicable:*

- *Remove all rules containing a default literal* $L = \sim A$ *such that* $A \in I$;
- *Remove from rules their default literals* $L = \sim A$ *such that* $\sim A \in I$;

and by next replacing all remaining default literals by proposition **u**.

[1]When the coherence principle is adopted, the truth value of L coincides with that of $(L, \sim\neg L)$. Taking programs in canonical form simplifies the techniques since we don't need to concern ourselves with objective literals in bodies in the modulo transformation, but only with default literals, just as for non–extended programs. The proof that generality is not lost can be found in [Alferes, 1993].

Programs $\frac{P}{I}p$ are by definition non–negative, and thus always has a unique Fitting–least 3–valued model, $least(\frac{P}{I}p)$, obtainable via a generalization of the van Emden–Kowalski least model operator Ψ [Przymusinska and Przymusinski, 1990]. In order to obtain all consequences of the program, even those leading to contradictions, as well as those arising from contradictions, we consider the consequences of all such possible $\frac{P}{I}p$ programs.

Definition 2.3 *Let* $QI = QT \cup \sim QF$ *be a set of literals. We define* $Coh^p(QI)$ *as the p–interpretation* $T \cup \sim F$ *such that* $T = QT$ *and* $F = QF \cup \{\neg L \mid L \in T\}$.

Definition 2.4 *Let* P *be an canonical extended logic program, I a p–interpretation, and let* $P_1, \ldots, P_i, \ldots$ *be all the permissible results of* $\frac{P}{I}p$. *Then:*

$$\Phi^p{}_P(I) = \bigcup_i Coh^p(least(P_i))$$

Definition 2.5 *The paraconsistent WFSX of an extended logic program* P, *denoted by* $WFSX_p(P)$, *is the least fixpoint of* Φ^p *applied to* P. *If some literal* L *belongs to* $WFSX_p(P)$ *we write* $P \models_p L$.

Indeed, it can be shown that Φ^p is monotonic, and therefore for every program it always has a least fixpoint, which can be obtained by iterating Φ^p starting from the empty set. It also can be shown that for a non–contradictory program P the paraconsistent WFSX coincides with WFSX.

Definition 2.6 *A program* P *is contradictory iff* $P \models_p \perp$.

To remove contradiction the first issue is defining which default literals $\sim A$ without rules, and so true by Closed World Assumption (CWA), may be **revised** to false, i.e. by adding A.

Example 2 *Consider* $P = \{a \leftarrow \sim b; \perp \leftarrow a\}$. $\sim b$ *is true by CWA on* b. *Hence, by the second rule, we have a contradiction. We argue the CWA may not be held of atom* b *as it leads to contradiction.*

Contradiction removal is achieved by adding to the original program P the complements of revisable literals:

Definition 2.7 (Revisables) *Let* $\mathcal{R}_P$ *be the set of all default literals* $\sim A$ *with no rules for* A *in an ELP* P. *The revisable literals of* P *are a subset of* $\mathcal{R}_P$. *A subset* S *of* $\sim\mathcal{R}_P$ *is a set of positive assumptions.*

Definition 2.8 (Revision of a program) *A set of positive assumptions* A *of* P *is a revision of* P *iff* $(P \cup A) \not\models_p \perp$

Example 3 *Consider the wobbly wheel problem:*

$$\begin{array}{rl} wobbly\_wheel \leftarrow & flat\_tyre \\ wobbly\_wheel \leftarrow & broken\_spokes \\ flat\_tyre \leftarrow & punctured\_tube \\ flat\_tyre \leftarrow & leaky\_valve \\ \perp \leftarrow & \sim wobbly\_wheel \end{array}$$

Using as revisables the literals $\sim broken\_spokes$, $\sim punctured\_tube$, and $\sim leaky\_valve$, there are 7 possible revisions, corresponding to the non–empty subsets of $\{punctured\_tube, broken\_spokes, leaky\_valve\}$.

Without loss of generality, as recognized in [Kakas and Mancarella, 1991], we can restrict ourselves to consider as revisables default literals for which there are no rules in the program; objective literals can always be made to depend on a default literal by adding a new rule or new literals in existing rules:

First, consider the case where a given literal is to be assumed true. For instance, in a diagnosis setting it may be wished to assume all components are ok unless it originates a contradiction. This is simply done, by introducing the rule $ok(X) \leftarrow \sim ab(X)$. Because $\sim ab(X)$ is true then $ok(X)$ also is. If $\sim ab(X)$ is revised $ok(X)$ becomes false, i.e. $ok(X)$ is revised from true to false.

Suppose now it is desirable to consider revisable an objective literal, say L, for which there are rules in the program. Let $\{L \leftarrow Body_1; \ldots; L \leftarrow Body_n\}$ be the rules in the definition of L. To make this literal "revisable" replace the rules for L by $\{L \leftarrow Body_1, \sim rev\_false(L,1); \ldots; L \leftarrow Body_n, \sim rev\_false(L,n); L \leftarrow rev\_true(L)\}$, with $\sim rev\_false(L,i)$ and $\sim rev\_true(L)$ being revisables. If L was initially false or undefined then to make it true it is enough to revise $\sim rev\_true(L)$. If L was true or undefined then to make it false it is sufficient to revise to false the literals $\sim rev\_false(L,i)$ in the bodies for L that are true or undefined.

In order to define the revision algorithm we'll need the concept of contradiction support. This notion will link the semantics' procedural and declarative aspects.

Definition 2.9 (Support set of a literal) *Support sets of any literal $L \in WFSX_p(P)$ of an ELP P, denoted by $SS(L)$, and are obtained as follows:*

1. *If L is a positive literal, then for each rule $L \leftarrow B_1, \ldots, B_n$ in P such that $P \models_p B_1, \ldots, B_n$, each $SS(L)$ is formed by the union of $\{L\}$ with some $SS(B_i)$ for each B_i.*
2. *If L is a default literal $\sim A$:*
   (a) *If no rules exist for A in P then $SS(L) = \{\sim A\}$.*
   (b) *If rules for A exist in P then choose from each rule with non-empty body a single literal whose complement belongs to $WFSX_p(P)$. For each such multiple choice there are several $SS(\sim A)$, each formed by the union of $\{\sim A\}$ with a SS of the complement of every chosen literal.*
   (c) *If $P \models_p \neg A$ then there exist,* additionally, *support sets SS of $\sim A$ equal to each $SS(\neg A)$.*

We are particularly interested in the supports of $\perp$, where the causes of contradiction can be found. The supports of $\perp$ in example 1 are $\{\sim b\}$ and $\{\sim c\}$. In examples 2 and 3 we have the single $SS(\perp) = \{\sim b\}$ and $SS(\perp) = \{\sim punctured\_tube, \sim broken\_spokes, \sim leaky\_valve\}$, respectively.

3 PREFERENCE LANGUAGE AND SEMANTICS

We have shown how to express nonmonotonic reasoning patterns using extended logic programming such as default reasoning and inheritance in [Pereira *et al.*, 1993a]. Additionally, we have discussed the relationship of our contradiction removal semantics to modelbased diagnosis and debugging in [Pereira *et al.*, 1993b, Pereira *et al.*, 1993c, Pereira *et al.*, 1993d]. However, while in these works we could easily express certain preference criteria (similar to those used in default reasoning) in our framework, other preference relations (as discussed for example in [Nejdl, 1991b, Nejdl and Banagl, 1992]) could not easily be represented.

To illustrate this, let us use an example from [Dressler and Bottcher, 1992]. We want to encode that we prefer a diagnosis including mode$_i$(C) to one including mode$_{i+1}$(C), where C is a component of the system to be diagnosed. The coding in default logic is a set of default rules of the form $\neg mode_1 \wedge \ldots \wedge \neg mode_i : mode_{i+1}/mode_{i+1}$, which can easily be translated into a logic program using WFSX semantics by including rules of the form $b_i(C) \leftarrow ab_1(C), \ldots, ab_{i-1}(C), \sim ab_i(C)$, where the b_i stand for the behaviour predicted by mode$_i$, and ab_i stands for the assumption that mode$_i$ has lead to a contradiction.

However, if we want to encode the slightly generalized preference relation, that prefers a diagnosis including mode$_i$(C1) to one including mode$_{i+1}$(C2) for any C1 and C2, this is no longer possible without enumerating all possible combinations of modes and components, which is not feasible in practice. Because of their declarative nature, logic programming and default logic give us a mechanism for prefering to include in an extension one fact over another (what we call *local preferences*), but not a mechanism for expressing *global preferences* stating we should only generate extensions including a certain fact after finding that no extensions including another fact exist, i.e. to attain sequencing of solutions.

To express global preferences, i.e. preferences over the order of revisions, we use a labeled directed acyclic and/or graph defined by rules of the form:

$$Level_0 \ll Level_1 \wedge Level_2 \wedge \ldots \wedge Level_n \;\; (n \geq 1) \quad (1)$$

$Level_i$ nodes in the graph are preference level identifiers. To each preference level node is associated a set of revisables denoted by $\mathcal{R}(Level_i)$. The meaning of a set of preference rules like (1) for some $Level_0$ is "I'm willing to consider the $Level_0$ revisions as "good" solutions (i.e the revisions of the original program using as revisables

$\mathcal{R}(Level_0)$) only if for some rule body its levels have been considered and there are no revisions at any of those levels." The root of the preference graph is the node denoted by $\mathtt{bottom}$, the bottom preference level. Thus, $\mathtt{bottom}$ cannot appear in the heads of preference rules. Additionally, there cannot exist a level identifier in the graph without an edge entering the node. This guarantees all the nodes are accessible from $\mathtt{bottom}$.

The revisions of the $\mathtt{bottom}$ preference level are (transitively) preferred to all other ones. Formally:

Definition 3.1 (Preferred Revisions) Let P be an ELP, and Π a preference graph containing a preference level Lev. The revision R is preferred wrt Π iff R is a minimal revision of P (in the sense of set inclusion), using revisables $\mathcal{R}(Lev)$, and there is an and-tree $\mathcal{T}$ embedded in Π, with root Lev, such that all leaves of $\mathcal{T}$ are **bottom** and no other preference levels in $\mathcal{T}$ have revisions.

Example 4 *Consider the following program P:*

$$
\begin{array}{ll}
have\_fun \leftarrow go(theater) & watch\_tv \leftarrow tv\_movie \\
have\_fun \leftarrow watch\_tv & watch\_tv \leftarrow tv\_show \\
have\_fun \leftarrow go(cinema) & \quad\bot \leftarrow \sim have\_fun \\
\neg go(theater) \leftarrow sold\_out(theater) & \\
\neg go(cinema) \leftarrow sold\_out(cinema) &
\end{array}
$$

P is contradictory because $\sim have\_fun$ is true. Its minimal revisions are $\{go(theater)\}$, $\{go(cinema)\}$, $\{tv\_movie\}$ and $\{tv\_show\}$ expressing have fun if I go to the theater or to the cinema, or stay at home watching a movie or tv show. If the next preference graph with associated revisables is added:

$$
\begin{array}{lll}
1 \ll \textbf{bottom} & 2 \ll \textbf{bottom} & 3 \ll 1 \wedge 2
\end{array}
$$
$$
\begin{array}{ll}
\mathcal{R}(\textbf{bottom}) = \{\sim go(theater)\} & \mathcal{R}(2) = \{\sim tv\_movie\} \\
\mathcal{R}(1) = \{\sim go(cinema)\} & \mathcal{R}(3) = \{\sim tv\_show\}
\end{array}
$$

then there is a unique preferred revision, namely $\{go(theater)\}$. Assume now the theater tickets sold out. We represent this situation by adding to the original program the fact $sold\_out(theater)$. Now the preferred revisions are $\{go(cinema)\}$ and $\{tv\_movie\}$. If the cinema tickets are also sold out the preferred revision will be $\{tv\_movie\}$.

I'll only stay at home watching the TV show if the cinema and theater tickets are sold out and there is no TV movie. If there is no TV show then I cannot remove contradiction, and cannot have fun. This constraint can be relaxed by replacing the integrity rule by $\bot \leftarrow \sim have\_fun, \sim sleep$ and adding an additional preference level with revisables $\{\sim sleep\}$ on top of 3. With this new encoding the preferred revision $\{sleep\}$ is produced.

Similarily, coming back to our example on preferences in a diagnosis system, we can encode the preference relation preferring diagnoses including a certain mode to ones not including it, by defining a linear order of levels where level i includes the set $\{ab_1(\_), \ldots ab_i(\_)\}$ as revisable literals in addition to the set of rules $b_i(C) \leftarrow ab_1(C), \ldots, ab_{i-1}(C), \sim ab_i(C)$.

This simple but quite general representation can capture the preference orderings among revisions described in the literature: minimal cardinality, most probable, minimal sets, circumscription, etc. Any of these preference orderings can be "compiled" to our framework. Furthermore, we can represent any preference based reasoning or revision strategy, starting from preferences which are just binary relations to preference relations, which are transitive, modular and/or linear ([Nejdl, 1991b, Kraus *et al.*, 1990]). Preferred revisions specified declaratively by preference level statements can be computed as follows:

Algorithm 3.1 (Preferred Revisions)
Input: An extended logic program P and a preference graph Π.
Output: The set of preferred revisions $PrefRev$.

$Explored = \{\}; ToExpl = \{\textbf{bottom}\};$
$PrefRev = \{\}; ContrLevels = \{\};$
repeat
  Let $lev \in ToExpl$; $Explored = Explored \cup \{lev\}$;
  $ToExpl = ToExpl - \{lev\}$;
  $LevelRev = MinRevisions(P, \mathcal{R}(lev))$;
  if $LevelRev \neq \{\}$ then
    $PrefRev = PrefRev \cup LevelRev$
  else $ContrLevels = ContrLevels \cup \{lev\}$
  Applicable = $\{l_0 \mid l_0 \ll l_1 \wedge \ldots \wedge l_n \in \Pi$ and $l_1, \ldots, l_n \in ContrLevels\}$
  $ToExpl = ToExpl \cup (Applicable - Explored)$
until $ToExpl = \{\}$

Algorithm 3.1 assumes the existence of a "magic" subroutine that given the program and the revisables returns the minimal revisions.

The computation of preferred revisions evaluates the preference rules in a bottom–up fashion, starting from the $\mathtt{bottom}$ level, activating the head of preference rule when all the levels in the body were unsuccessfully tried. At each new activated level the revision algorithm is called. If there are revisions then they are preferred ones, but further ones can be obtained. If there are no revisions, we have to check if new preference rules are applicable, bygenerating new active levels. This process is iterated till all active levels are exhausted.

4 EXAMPLES OF APPLICATION

We've tested REVISE on several examples, including the important problem of representing diagnostic strategies in a modelbased reasoning system. Below are two examples.

4.1 Two Inverter Circuit

In this example we present two extended diagnosis cases which illustrate the use of the preference graph to capture diagnosis strategies. Consider the simple two inverter circuit in figure 1.

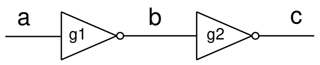

Figure 1: Two Inverter Circuit

The normal and abnormal behaviour of the inverter gate is modelled by the rules below. We assume that our inverter gates have two known modes of erroneous comportment, either the output is always "0" (mode "stuck at 0") or is always "1" (mode "stuck at 1"). The fault mode "unknown" describes unknown faults, with no predicted behaviour. The last argument, T, is a time–stamp that permits the modelling of distinct observations in time. It is also implicit that an abnormality is permanent.

$$inv(G, I, 1, T) \leftarrow \sim ab(G), node(I, 0, T)$$
$$inv(G, I, 0, T) \leftarrow \sim ab(G), node(I, 1, T)$$

$$inv(G, \_, 0,_{)} \leftarrow ab(G), s\_at\_0(G)$$
$$inv(G, \_, 1,_{)} \leftarrow ab(G), s\_at\_1(G)$$

$$s\_at\_0(G) \leftarrow fault\_mode(G, s0)$$
$$s\_at\_1(G) \leftarrow fault\_mode(G, s1)$$
$$unknown(G) \leftarrow fault\_mode(G, unknown)$$

The connections among components and nodes are described by:

$$node(b, B, T) \leftarrow inv(g1, a, B, T)$$
$$node(c, C, T) \leftarrow inv(g2, b, C, T)$$

The first integrity rule below ensures that the fault modes are exclusive, i.e. to an abnormal gate at most one fault mode can be assigned. The second one enforces the assignment of at least one fault mode to anomalous components. The last integrity rule expresses the fact that to each node only one value can be predicted or observed at a given time.

$$\perp \leftarrow fault\_mode(G, S1), fault\_mode(G, S2), S1 \neq S2$$
$$\perp \leftarrow ab(G), \sim fault\_mode(G, s0), \sim fault\_mode(G, s1), \sim fault\_mode(G, unknown)$$
$$\perp \leftarrow node(X, V1, T), node(X, V2, T), V1 \neq V2$$

Now we show how the preference graph can be used to implement distinct reasoning modes. The basic idea is to focus reasoning by concentrating on probable failures first (simple views, high abstraction level, etc...), to avoid reasoning in too large a detail. In this example, we'll prefer single faults to multiple faults (i.e. more than one component is abnormal), fault mode "stuck at 0" to "stuck at 1" and the latter to the "unknown" fault mode. One possible combination of these two preferences is expressed using the following integrity rules and preference graph. This graph and its associated revisables are depicted in figure 2.

$$\perp \leftarrow fault\_mode(\_, s1), \sim s0\_impossible$$
$$\perp \leftarrow fault\_mode(\_, unknown), \sim s0\_impossible$$
$$\perp \leftarrow fault\_mode(\_, unknown), \sim s1\_impossible$$
$$\perp \leftarrow fault\_mode(G1, \_), fault\_mode(G2, \_) \; G2 \neq G1, \sim single\_fault\_impossible$$

1 $\ll$ **bottom**   2 $\ll$ **bottom**
3 $\ll$ 1   4 $\ll$ 1 $\wedge$ 2
5 $\ll$ 3 $\wedge$ 4

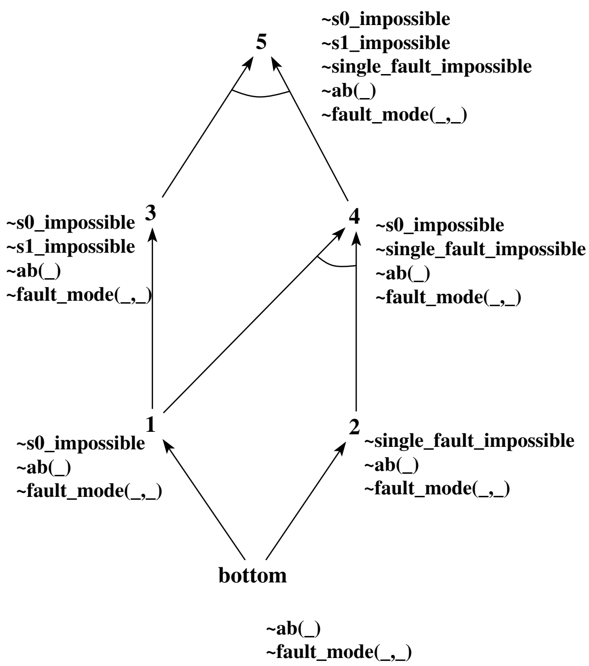

Figure 2: Reasoning Modes Preference Graph

In the bottom level only $\sim ab(\_)$ and $\sim fault\_mode(\_, \_)$ are revisables. Because neither of $\sim s0\_impossible$, $\sim s1\_impossible$ and $\sim single\_fault\_impossible$ are revisables, the integrity rules enforce single "stuck at 0" faults. Level 1 and level 3 correspond to single "stuck at 1" faults and single "unknown faults." Level 2 express possible multiple "stuck at 0" faults. Level 4 captures multiple "stuck at 0" or "stuck at 1" faults. Finally, all kind of faults, single or multiple, are dealt with in level 5.

We could migrate the knowledge embedded in the last four previous integrity rules to the preference graph. This preference graph is more elaborate (but probably more intuitive) as shown in figure 3. The dashed boxes correspond to the levels of the previous preference graph and are labeled accordingly. This demonstrates how the meta–knowledge represented in the preference graph could move to the program, with a substancial reduction of the preference relation. If instead of 2 components we had 100, the extended graph we will have about 304 nodes and the smaller one is identical to the one of figure 2. The general conditions of the preference graph that allow this transference of information are the subject of further investigation.

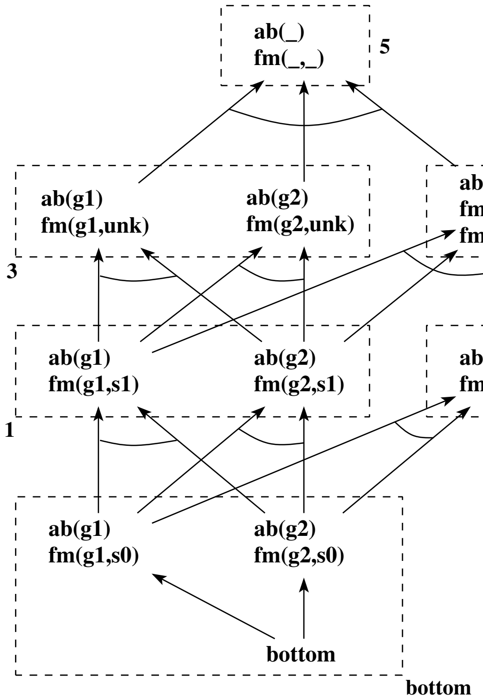

Figure 3: Extended Preference Graph

Suppose that in the first experiment made the input is 0 and the values at nodes b and c are also 0 (see figure 4). These observations are modelled by the facts:

$$node(a, 0, t0) \quad node(b, 0, t0) \quad node(c, 0, t0)$$

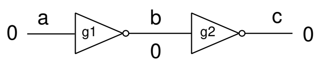

Figure 4: First Experiment

This program is contradictory with the following single preferred revision obtained at level 2:

$$\{ab(g1), ab(g2), fault\_mode(g1, s0), fault\_mode(g2, s0), single\_fault\_impossible\}$$

The facts $\{node(a, 0, t1),\ node(b, 1, t1),\ node(c, 0, t1)\}$ describe the results of a second experiment made in the circuit. The above two experiments together generate, at level 5, the revisions below. Notice that the second one is at first sight non intuitive. We formulated the problem employing the consistency–based approach to diagnosis, therefore it is consistent with the observations to have both gates abnormal with "unknown" faults.

$$\{ \ s0\_impossible, s1\_impossible, single\_fault\_impossible, ab(g1), fault\_mode(g1, unknown), ab(g2), fault\_mode(g2, s0) \ \}$$

$$\{ \ s0\_impossible, s1\_impossible, single\_fault\_impossible, ab(g1), fault\_mode(g1, unknown), ab(g2), fault\_mode(g2, unknown) \ \}$$

4.2 MULTIPLE MODELS AND HIERARCHY

In the rest of this section we will consider a small abstract device. We will focus on the use of different abstraction and diagnosis strategies and leave out the actual details of the models.

This device is built from chips and wires and has a certain behaviour. If it does not show that behaviour, then we consider one of its parts abnormal.

% concept: structural hierarchy,
% axiom: some part is defect, iff one its subparts is defect,
% strategy: decompose the system into its subcomponents

$$\neg a(device) \leftarrow {\sim}a(chips), {\sim}a(wires)$$
$$behaviour(device) \leftarrow {\sim}a(device)$$

% first observation
% contradiction found at highest level

$$\neg behaviour(device)$$

In our case, either the chips can be faulty or the wires can be faulty. To check that, we use a functional model if available (in the case of the chips) or a physical model. As we will see later, our specified preference order leads us to suspect the chips first.

% concept: functional/physical hierarchy.
% a contradiction is found, if the functional model leads to
% contradiction. If this is the case, check the physical models
% of the suspected component (these axioms come later)

$$\neg a(chips) \leftarrow {\sim}a(functionalChipModel)$$
$$assump(ChipModel) \leftarrow a(FunctionalChipModel)$$

% no functional model for wires,
% directly start with the physical model

$$\neg a(wires) \leftarrow {\sim}a(physicalWireModel)$$

When testing the chips, we use the functional hierarchy and get three function blocks fc1, fc2 and fc3.

% enumeration of possible abnormal
% components (functional decomposition)

$$\neg a(functionalChipModel) \leftarrow {\sim}af(fc1), {\sim}af(fc2), {\sim}af(fc3)$$

The functions of these blocks and their interdependencies are described in an arbitrary way, below we show a very simple sequential relationship.

% behaviour of single components in one model view
% simple kind of sequential propagation, model valid
% only if we just use chip model (no faulty wires)

$$\begin{aligned} fb(fc1) &\leftarrow \sim af(fc1), assump(ChipModel) \\ fb(fc2) &\leftarrow \sim af(fc2), fb(fc1), \\ &\quad assump(ChipModel) \\ fb(fc3) &\leftarrow \sim af(fc3), fb(fc2), \\ &\quad assump(ChipModel) \\ behaviour(device) &\leftarrow fb(fc3), assump(ChipModel) \end{aligned}$$

We really find out, that the functional block fc1 malfunctions. Now we have to check, which physical components compose that functional block and which one of them is faulty. As functional block fc1 consists of two physical components c1 and c4, these two are our suspects after the first two observations.

% second observation, restricts the found diagnoses to fc1

$\neg fb(fc1)$

% concept: multiple models.
% When a fault is found using the functional model, check the
% physical chip models, prefer physical chip model 1 over
% physical chip model 2.

type(c1,chip)type(c2,chip)
type(c3,chip)type(c4,chip)

The functional decomposition need not be the same as the physical composition, functional component c1 consists of physical components c1 and c4), physical model 2 (ap2) corresponds to the unknown failure mode, we prefer physical mode 1, if it resolves the contradiction.

$$\begin{aligned} &\neg af(fc1) \leftarrow \sim ap(c1), \sim ap2(c1), \sim ap(c4), \sim ap2(c4) \\ &\neg af(fc2) \leftarrow \sim ap(c2), \sim ap2(c2) \\ &\neg af(fc3) \leftarrow \sim ap(c3), \sim ap2(c3) \end{aligned}$$

On the level of physical components we have descriptions of behavioural models, in our case models of the correct mode and two fault modes. We further have the assumption, that there is no other unknown fault mode.

% behavioue single components in the physical model 1

$$\begin{aligned} &pb(c1) \leftarrow \sim ap(c1), \sim ap2(c1), assump(ChipModel) \\ &pb(c2) \leftarrow \sim ap(c2), \sim ap2(c2), assump(ChipModel) \\ &pb(c3) \leftarrow \sim ap(c3), \sim ap2(c3), assump(ChipModel) \\ &pb(c4) \leftarrow \sim ap(c4), \sim ap2(c4), assump(ChipModel) \end{aligned}$$

% exclusive failure modes for chips in physical model 1

$\neg ap(C) \leftarrow \sim f1(C), \sim f2(C), type(C, chip)$

% behaviour of components with fault mode 1

$faultb1(C) \leftarrow f1(C)$

% behaviour of components with fault mode 2

$faultb2(C) \leftarrow f2(C)$

Now, we have two more observations, which tell us, that c1 is not functioning according to its fault mode 1 and c4 is not either. So, as we prefer fault mode 1 to fault mode 2 (as specified later in the preference ordering) the effect of the third observation is to focus on c4 in fault mode 1, while the effect of the fourth observation is to reconsider both c4 and c1 in fault mode 2.

% third observation

$\neg faultb1(c1)$

% fourth observation

$\neg faultb1(c4)$

Finally, with a last observation, we restrict our suspect list again to c4 in fault mode 2. Finally, observation 6 leads us to consider unknown faults.

% fifth observation

$\neg faultb2(c1)$

% sixth observation

$\neg faultb2(c4)$

For the wires, we just have a physical model:

% types

$type(w1, wire)$
$type(w2, wire)$

$\neg a(physicalWireModel) \leftarrow \sim ap(w1), \sim ap(w2)$

$pb(w1) \leftarrow \sim ap(w1)$
$pb(w2) \leftarrow \sim ap(w2)$

Finally, we specify a specific diagnosis strategy by using the preference order of revisables. The preference graph below starts to tes the most abstract hierarchy level (**bottom**). If something is wrong, then go to the next hierarchy level (chips and wires) on this level, first consider chip faults with fault mode 1 first (level 1). Otherwise try single fault with fault mode 2 (level 2). If this doesn't explain the behaviour try a double fault with fault mode 1 or a single unknown fault (3,4). Otherwise suspect faulty wires (level 5). If this does not work then everything is possible in the last level (6).

$1 \ll$ **bottom**	$2 \ll 1$
$3 \ll 2$	$4 \ll 2$
$5 \ll 3 \wedge 4$	$5 \ll 6$

The sequence of revisions and the revisable levels are listed below:

Level	Revisions
bottom	$\{a(device)\}$
1	$\{a(device), a(chips), ab(\_), af(\_), ap(\_), f1(\_), a(functionalChipModel)\}$
2	$\{a(device), a(chips), ab(\_), af(\_), ap(\_), f2(\_), a(functionalChipModel)\}$
3	$\{a(device), a(chips), ab(\_), af(\_), ap(\_), ap2(\_), a(functionalChipModel)\}$
4	$\{a(device), a(chips), ab(\_), a(functionalChipModel), af(\_), not\_single\_fault, ap(\_), f1(\_)\}$
5	$\{a(device), a(wires), ab(\_)), af(\_), ap(\_), a(physicalWireModel)\}$
6	$\{a(device), a(chips), a(functionalChipModel), ab(\_)), af(\_), ap(\_), ap2(\_), f1(\_), f2(\_), a(wires), a(physicalWireModel)\}$

Table 1: Levels and Revisables

Observation 1, Revisions at level 1:

$\{a(chips), a(device), a(functionalChipModel), af(fc1), ap(c4), f1(c4)\}$
$\{a(chips), a(device), a(functionalChipModel), af(fc1), ap(c1), f1(c1)\}$
$\{a(chips), a(device), a(functionalChipModel), af(fc2), ap(c2), f1(c2)\}$
$\{a(chips), a(device), a(functionalChipModel), af(fc3), ap(c3), f1(c3)\}$

Observation 2, Revisions at level 1:

$\{a(chips), a(device), a(functionalChipModel), af(fc1), ap(c4), f1(c4)\}$
$\{a(chips), a(device), a(functionalChipModel), af(fc1), ap(c1), f1(c1)\}$

Observation 3, Revisions at level 1:

$\{a(chips), a(device), a(functionalChipModel), af(fc1), ap(c4), f1(c4)\}$

Observation 4, Revisions at level 2:

$\{a(chips), a(device), a(functionalChipModel), af(fc1), ap(c4), f2(c4)\}$
$\{a(chips), a(device), a(functionalChipModel), af(fc1), ap(c1), f2(c1)\}$

Observation 5, Revisions at level 2:

$\{a(chips), a(device), a(functionalChipModel), af(fc1), ap(c4), f2(c4)\}$

Observation 6, Revisions at level 3:

$\{a(chips), a(device), a(functionalChipModel), af(fc1), ap2(c1)\}$
$\{a(chips), a(device), a(functionalChipModel), af(fc1), ap2(c4)\}$

5 REVISION ALGORITHM

The main motivation for this algorithm is that at distinct preference levels the set of revisables may differ, but it is necessary to reuse work as much as possible, or otherwise the preferred revisions algorithm will be prohibitively inefficient. We do so by presenting an extension to Reiter's algorithm [Reiter, 1987], corrected in [Greiner *et al.*, 1989], that fully complies with this goal. The preference revision process can be built upon this algorithm simply by calling it with the current set of revisables, as shown in section 3. This permits the reuse of the previous computations that matter to the current set of revisables. The algorithm handles arbitrary sets of assumptions, handles different sets of revisables at different levels, and provides a general caching mechanism for the computation of minimal "hitting sets" obtained at previous levels.

The main data structure is a DAG constructed from a set $\mathcal{C}$ of conflict sets (all the inconsistencies) and a non-contradictory initial context Cx which expresses what are the initial truth values of the revisables. An inconsistency is a support of contradiction. In our setting, these conflict sets are the intersection of a support set of $\perp$ with the set of revisables and their negations. Thus, a conflict set contains only literals that were already revised and literals that can be revised, which jointly lend support ot contradiction.

Remark that the next algorithm is described independently of the initial context and of the type of revisable literals (they may be true positive literals that are allowed to become false through revision).

A generalized–hitting–set directed–acyclic–graph (*GHS-DAG*) is a DAG where nodes are labeled with sets of conflict sets. Edges are labeled with negations of revisables. In the basic algorithm (without pruning) the need for labeling nodes with sets of conflict sets is not evident. For clarity we'll describe the basic algorithm assuming that there is only one conflict set in the label. After introducing pruning we'll slightly revise the basic algorithm to cope with the more general case.

In the following, D_{new}, is used to build the level specific GHS–DAG using the conflict sets stored in the GHS–DAG D (used as the global cache). While computing the level specific DAG the global cache is updated accordingly. This latter GHS–DAG will be fed into the next iteration of algorithm 3.1 to minimize computation effort in less preferred levels. The minimal, level specific, revisions for a given set of revisables are obtained from D_{new}. Conflict sets are returned by a theorem prover; as in [Reiter, 1987, Greiner *et al.*, 1989] the algorithm tries to minimize the number of calls to the theorem prover.

The set of all revisables is a non-contradictory set of literals denoted by $\mathcal{R}$. Intuitively, if a literal L (resp. $\sim L$) belongs to $\mathcal{R}$ then we allow L to change its truth-value to false (resp. to true). A conflict set CS is a subset of $\mathcal{R}$. A conflict set contains literals that cannot be simultaneously true (similar to the definition of NOGOOD [McDermott, 1991]). A context is a subset of $\mathcal{R} \cup \sim\mathcal{R}$.

The reader is urged to recourse to figure 5 when reading the basic algorithm below, where the initial context is $\{a, b, c, d\}$ and $Rev = \{\sim a, \sim b, \sim c, \sim d\}$. The set of inconsistencies, $\mathcal{C}$, contains the sets appearing in the figure. We assume, for simplicity, that the global GHS–DAG is empty and therefore in every step D and D_{new} are equal. The nodes are numbered by order of creation.

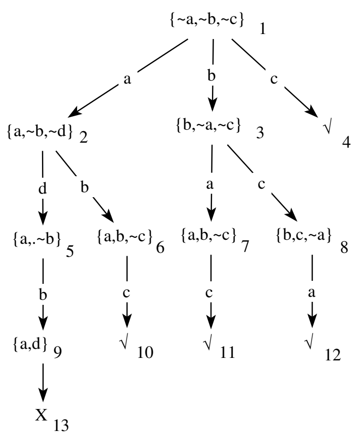

Figure 5: Basic Algorithm

0. Let $\mathcal{C}$ be a set of conflict sets and Cx context. The first time this algorithm is called GHS–DAG D is empty.
1. Let $Rev \subseteq \mathcal{R}$ be a set of revisables, D a GHS-DAG and D_{new} an empty GHS-DAG. If D is not empty then insert a clone of its root node in D_{new}. Otherwise, insert in D_{new} an unlabeled node and a clone node in D.
2. Process the nodes in D_{new} in breadth–first order. Let n be a node. The set $H(n)$ contains the revisions of literals in the path from the root to the node and $M(n)$ is the context at node n. To process node n :
   (a) Define $H(n)$ to be the set of edge labels on the path in D_{new} from the root down to node n (empty if n is the root). Let $M(n) = \left[Cx - \sim H(n)\right] \cup H(n)$.
   (b) If n is not labeled then if for all $X \in \mathcal{C}$, $X \not\subseteq M(n)$ then label n and its clone with the special symbol $\surd$. Otherwise, label the node with one arbitrary X such that $X \subseteq M(n)$. Copy the label to its clone in D.
   If n is already labeled denote its label by X.
   If $X \cap Rev = \{\}$ then mark the node in D_{new} as closed.
   (c) If n is labeled by a set $X \in \mathcal{C}$, then for each $l_i \in X \cap Rev$, generate a new outgoing arc labeled by $\sim l_i$ leading to a new node m_i. If an arc already exists in D, from a clone of n and label $\sim l_i$ to some node r_i, then copy the label of r_i to m_i in D_{new}. Node r_i becomes the clone of m_i. Otherwise m_i remains unlabeled and a clone arc is created in D.
3. Return the resulting GHS–DAGs, D_{new} and D.

If the node is not labeled in step 2b then it was unexplored. The algorithm attempts to find a conflict set compatible with the current context. If there is no conflict then a revision is found (possibly not minimal) and it is marked with $\surd$. If a conflict set is found then it is intersected with the current revisables. If the intersection is empty a non-revisable contradiction was found. Otherwise, the node must be expanded in order to restore consistency. This is accomplished by complementing a single revisable literal in the conflict set. This process is iterated till there are no more nodes to expand.

An access to the set of inconsistencies in step 2b can be regarded has a request to the theorem prover to compute another conflict set. Also, notice in the end D_{new} will be contained in D. All the revisions wrt the specified set of revisables can be obtained from D_{new}. We now define pruning enhancements to D_{new} such that, in the end, the minimal revisions can be obtained from this new DAG.

The result of the application of the next optimizations to the problem described in figure 5 is shown in figure 6. Node 5 is obtained by application of pruning step 3a. Node 6 is reused by node 3 and nodes 7 and 8 were closed by node 4.

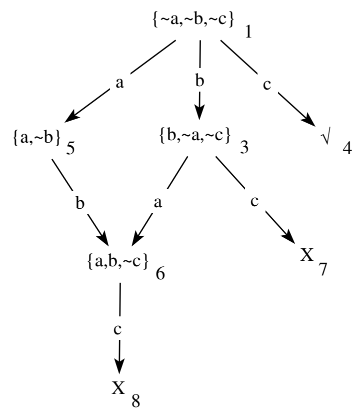

Figure 6: Pruning Enhancements

1. Reusing nodes: this algorithm will not always generate a new node m_i as a descendant of node n in step 2c:
   (a) If there is a node n' in D_{new} such that $H(n') = H(n) \cup \{l\}$, then let the l–arc under n point to this existing node n', and reflect this in its clone in D. Thus, these nodes will have more than one parent.
   (b) Otherwise generate m_i as described in the basic algorithm.

2. Closing: If there is a node m in D_{new} labeled with $\surd$ and $H(m) \subseteq H(n)$, then close node n and its clone in D.
3. Pruning: Let X be the label of node n in step 2b, and m be another node in D_{new} labeled with conflict set Y. Attempt to prune D_{new} and D as follows:
   (a) Pruning D_{new} : If $X \cap Rev \subseteq Y \cap Rev$ and $X \cap H(n) \subseteq H(m)$, then relabel m with X. For any $\sim l_i$ in $(Y - X) \cap Rev$ the l_i–arc under m is no longer allowed. The node connected by this arc and all of its descendants are removed, except for those nodes with another ancestor which is not being removed. This step may eliminate the node currently being processed.
   (b) Pruning D : Let n' and m' be the clones of n and m in D, respectively.
      i. If $X \cap \mathcal{R} \subseteq Y \cap \mathcal{R}$ and $X \cap H(n') \subseteq H(m')$, for some Y in m'. Apply the procedure in 3a) to D, taking into account that more than one set may label the node: remove all the Ys that verify the condition, insert X in the label and retain l–edges and descendant nodes such that $\sim l$ belongs to some of the remaining sets. Notice this procedure was already applied to D_{new}.
      ii. If the condition in 3a) was verified by n and m then insert in the label of m' the set X.

The reuse and closing of nodes avoids redundant computations of conflict sets. The closing rule also ensures that in the end the nodes labeled with $\surd$ are the minimal revisions. The pruning strategy is based on subset tests, as in Reiter's original algorithm, but extended to take into account the contexts and revisables. The principle is simply that given two conflict sets one contained in the other, the bigger one can be discarded when computing minimal revisions. The algorithm handles the case when a conflict set is a subset of another wrt the revisables, but not if the revisables are not considered. Also notice that condition 3(b)ii is redundant if step 3(b)i was applied.

The need to consider nodes labeled by several conflict sets is justified by the pruning step 3(b)ii: the set X is minimal wrt the current set of revisables. We store this set in the global cache D for later use by the algorithm. To reap the most benefit from this improvement it is necessary to change step 2b) by introducing the following test:

> "If n is labeled by conflict sets, i.e. it was previously computed, then intersect these sets with Rev. Select from the obtained sets a minimal one such that the number of nodes below this node, wrt Rev, is maximum. Relabel n with this set."

The heuristic in the modified step is to use a minimal set wrt to the current set of revisables having the most of search space explored. This has two main advantages: first, the new algorithm can ignore redundant branches (the minimal set condition); second it directs the search to the sub-DAG with less nodes remaining to be explored, reusing maximally work done before.

It can be shown that if the number of conflict sets is finite, and they themselves are finite, then this algorithm computes the minimal revisions and stops. It is known this problem is NP-Hard: an exponential number of minimal revisions can be generated. Under the previous conditions algorithm 3.1 is sound and complete, if it has a finite number of preference rules (wrt def. 3.1).

The computation of the preferred revisions of example in section 4.1 is portrayed in figure 7, with the obvious abreviations for the literals appearing. The whole example has 25 minimal inconsistencies, i.e. minimal support sets of $\perp$. The computations done at each level are bounded by the labeled bold lines. The dashed line represents the bound of the level specific GHS–DAG at levels **bottom**, 2 and 4. These leves all nodes are closed, i.e. there are no revisions. Notice that pruning steps 3a) and 3(b)ii were applied between the node initially labeled with $\{\sim ab(g2)\}$ and the nodes labeled with $\{\sim fm(g1, uk), \sim s0i\}$ and $\{fm(g1, uk), \sim s1i\}$.

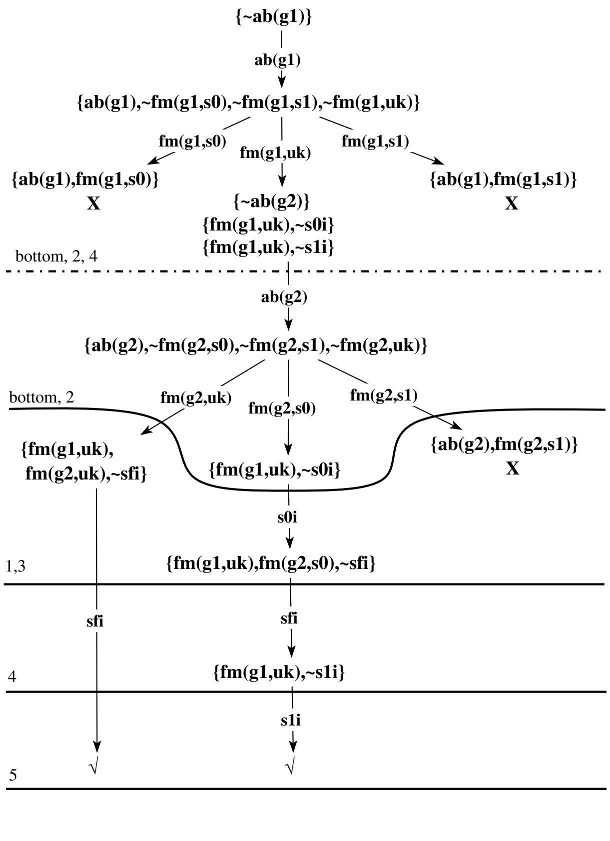

Figure 7: Preferred Revisions Computation

As the reader may notice, the higher the level is in the preference graph the deeper is the GHS–DAG generated. This corresponds to what is desired: preferred revisons should be obtained with less theorem proving costs.

The efficiency of the algorithm is highly dependent on the order in which conflict sets are generated. It is possible to devise very nasty examples having $n+1$ conflict sets with a single revision of cardinality 1, where the above algorithm has to explore and generate an exponential GHS–DAG. One possible solution to this problem is to guarantee that the theorem prover only returns minimal sets of inconsistencies. This can be done by an intelligent search strategy used by the theorem prover. This topic will be the subject of a future paper.

6 COMPARISONS AND CONCLUSIONS

We start by concentrating on diagnosis strategies, only recently approached in [Struss, 1989, Dressler and Bottcher, 1992, Böttcher and Dressler, 1994], where they discuss the necessity of including diagnostic strategies in a modelbased reasoning system such as when to use structural refinement, behavioural refinement, how to switch to different models, and when to use which simplifying assumptions. [Dressler and Bottcher, 1992] also gives an encoding of these preferences on top of an NM-ATMS, using both default rules of the form discussed earlier for local preferences and meta-level rules for explicitly specifying global preference during the revision process.

The disadvantage of the implementation discussed in [Dressler and Bottcher, 1992] and its extended version is that the specification of these preferences is too closely intertwined with their implementation (NM-ATMS plus belief revision system on top) and that different preferences must be specified with distinct formalisms. Default logic allows a rather declarative preferences specification, while the meta-rules of [Dressler and Bottcher, 1992] for specifying global preferences have a too procedural flavour.

In contrast, REVISE builds upon a standard logic programming framework extended with the explicit specification of an arbitrary (customizable) preference relation (including standard properties ([Kraus *et al.*, 1990, Nejdl, 1991b]) as needed). Expression of preferences is as declarative as expressing diagnosis models, uses basically the same, formalism and is quite easy to employ according to our experience.

In our diagnosis applications, although our implementation still has some efficiency problems due to being implemented as a meta-interpreter in Prolog (a problem which can be solved without big conceptual difficulties), we are very satisfied with the expressiveness of REVISE and how easy it is to express in it a range of important diagnosis strategies.

We are currently investigating how to extend our formalism for using an order on sets of diagnoses/sets of worlds/sets of sets of assumption sets, instead of the current order on diagnoses/worlds (or assumption sets). Without this greater expressive power we are not able to spell out strategies depending on some property out of the whole set of diagnoses (two or three of such strategies are discussed in a belief revision framework in [Böttcher and Dressler, 1994]).

Revising a knowledge base has been the main topic for belief revision systems and it is quite interesting to compare REVISE to a system like IMMORTAL [Chou and Winslett, 1991]. IMMORTAL starts from a consistent knowledge base (not required by our algorithm) and then allows updating it. These updates can lead to inconsistencies and therefore to revisions of the knowledge base. In contrast to our approach, all inconsistencies are computed statically before any update is done, which for larger examples is not feasible, in our opinion. Also, due to their depth-first construction of hitting sets, they generate non-minimal ones, which have to be thrown away later. (This could be changed by employing a breadth-first strategy.)

Priority levels in IMMORTAL are a special kind of our preference levels, where all levels contain disjoint literals and the levels are modular, both of which properties can be relaxed in REVISE (and have to be in some of our examples). Accordingly, IMMORTAL does not need anything like our DAG global data structure for caching results between levels.

Additionally, IMMORTAL computes all conflict sets and then removes non-updateable literals, in general leading to duplicates; therefore, in our approach, we have less conflict sets and so a faster hitting sets procedure (and less theorem prover costs).

Comparing our work to database updates and inconsistency repairing in knowledge bases, we note that most of this work takes some kind of implicit or explicit abductive approach to database updates, and inconsistency repairs are often handled as a special case of database update from the violated contraints (such as in [Wüthrich, 1993]). If more than one constraint is violated this leads to a depth-first approach to revising inconsistencies which does not guarantee minimal changes. Moreover, we do not know of any other work using general preference levels. All work which uses some kind of priority levels at all uses disjoint priority levels similar to the IMMORTAL system.

Compared to other approaches, which are based on specific systems and specific extensions of these systems such as [Dressler and Bottcher, 1992], REVISE has big advantages in the declarativity, and is built upon on a sound formal logic framework.

We therefore believe the REVISE system is an appropriate tool for coding a large range of problems involving revision of knowledge bases and preference relations over them. REVISE is based on sound logic programming semantics, described in this paper, and includes some interesting implementation concepts exploiting relationships between belief revision and diagnosis already discussed in very preliminary form in [Nejdl, 1991a]. While REVISE is still a prototype, we are currently working on further improving its efficiency, and are confident the basic implementation structure of the current system provides an easily extensible backbone for efficient declarative knowledge base revision systems.

Acknowledgements

We thank project ESPRIT Compulog 2, JNICT Portugal and the Portugal/Germany INIDA program for their support.

References

[Alferes, 1993] José Júlio Alferes. *Semantics of Logic Programs with Explicit Negation*. PhD thesis, Universidade Nova de Lisboa, October 1993.

[Böttcher and Dressler, 1994] Claudia Böttcher and Oskar Dressler. Diagnosis process dynamics: Putting diagnosis on the right track. *Annals of Mathematics and Artificial Intelligence*, 11(1–4), 1994. Special Issue on Principles of Model-Based Diagnosis. To appear.

[Chou and Winslett, 1991] Timothy S-C Chou and Marianne Winslett. Immortal: A model-based belief revision system. In *In Proc. KR'91*, pages 99–110, Cambridge, April 1991. Morgan Kaufmann Publishers, Inc.

[de Kleer, 1991] Johan de Kleer. Focusing on probable diagnoses. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 842–848, Anaheim, July 1991. Morgan Kaufmann Publishers, Inc.

[Dressler and Bottcher, 1992] O. Dressler and C. Bottcher. Diagnosis process dynamics: Putting diagnosis on the right track. In *Proc. 3nd Int. Workshop on Principles of Diagnosis*, 1992.

[Friedrich and Nejdl, 1992] Gerhard Friedrich and Wolfgang Nejdl. Choosing observations and actions in model-based diagnosis-repair systems. In *In Proc. KR'92*, pages 489–498, Cambridge, MA, October 1992. Morgan Kaufmann Publishers, Inc.

[Gelder *et al.*, 1991] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[Greiner *et al.*, 1989] Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.

[Hamscher *et al.*, 1992] W. Hamscher, L. Console, and J. de Kleer. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, 1992.

[Kakas and Mancarella, 1991] A.C. Kakas and P. Mancarella. Generalized stable models: A semantics for abduction. In *Proc. ECAI'90*, pages 385–391, 1991.

[Kraus *et al.*, 1990] Sarit Kraus, Daniel Lehmann, and Menachem Magidor. Nonmonotonic reasoning, preferential models and cumulative logics. *Artificial Intelligence*, 44(1–2):167–207, 1990.

[Lackinger and Nejdl, 1993] Franz Lackinger and Wolfgang Nejdl. DIAMON: A model-based troubleshooter based on qualitative reasoning. *IEEE Expert*, February 1993.

[McDermott, 1991] Drew McDermott. A general framework for reason maintenance. *Artificial Intelligence*, 50(3):289–329, 1991.

[Nejdl and Banagl, 1992] Wolfgang Nejdl and Markus Banagl. Asking about possibilities — revision and update semantics for subjunctive queries. In *In Proc. KR'92*, Cambridge, MA, October 1992. Also appears in the ECAI-Workshop on Theoretical Foundations of Knowledge Representation and Reasoning, August 1992, Vienna.

[Nejdl, 1991a] Wolfgang Nejdl. Belief revision, diagnosis and repair. In *Proceedings of the International GI Conference on Knowledge Based Systems*, München, October 1991. Springer-Verlag.

[Nejdl, 1991b] Wolfgang Nejdl. The P-Systems: A systematic classification of logics of nonmonotonicity. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 366–372, Anaheim, CA, July 1991.

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

[Pereira *et al.*, 1993a] L. M. Pereira, J. N. Aparício, and J. J. Alferes. Non–monotonic reasoning with logic programming. *Journal of Logic Programming. Special issue on Nonmonotonic reasoning*, pages 227–263, 1993.

[Pereira *et al.*, 1993b] L. M. Pereira, C. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal. In L. M. Pereira and A. Nerode, editors, *2nd Int. Ws. on Logic Programming and NonMonotonic Reasoning*, pages 316–330. MIT Press, 1993.

[Pereira *et al.*, 1993c] L. M. Pereira, C. V. Damásio, and J. J. Alferes. Debugging by diagnosing assumptions. In *Automatic Algorithmic Debugging, AADEBUG'93*. Springer–Verlag, 1993.

[Pereira *et al.*, 1993d] L. M. Pereira, C. V. Damásio, and J. J. Alferes. Diagnosis and debugging as contradiction removal in logic programs. In *Proc. 6th Portuguese Conference On Artificial Intelligence (EPIA'93)*. Springer–Verlag, LNAI 727, 1993.

[Przymusinska and Przymusinski, 1990] H. Przymusinska and T. Przymusinski. Semantic issues in deductive databases and logic programs. In R. Banerji, editor, *Formal Techniques in AI, a Sourcebook*, pages 321–367. North Holland, 1990.

[Reiter, 1987] R. Reiter. A theory of diagnosis from first principles. *AI*, 32:57–96, 1987.

[Struss, 1989] P. Struss. Diagnosis as a process. In *Working Notes of First Int. Workshop on Model-based Diagnosis*, Paris, 1989. Also in [Hamscher *et al.*, 1992].

[Wüthrich, 1993] B. Wüthrich. On updates and inconsistency repairing in knowledge bases. In *Proceedings of the International IEEE Conference on Data Engineering*, Vienna, Austria, April 1993.