Diagnosis and Debugging as Contradiction Removal

Luís Moniz Pereira
Carlos Viegas Damásio
José Júlio Alferes
CRIA Uninova and DCS, U.Nova de Lisboa
2825 Monte da Caparica, Portugal
({lmp,cd,jja}@fct.unl.pt)

1 Introduction

Recent approaches make use of logic programming (LP), and in particular LP with explicit negation (extended logic programming–XLP) [13, 8, 9], to solve and represent nonmonotonic reasoning problems [20, 17]. The aim of this paper is to enlarge in an unified way the scope of XLP applications to diagnosis, and to declarative debugging. The expressive power of XLP to do so is attained by allowing would be contradictory programs to be adequately revised by a contradiction removal semantics which withdraws assumptions that support contradiction and revises them to false.

We elaborate on the work of [15, 16] on contradiction removal of extended logic programs (CRSX), and also show how Reiter's algorithm DIAGNOSE [25, 10] is used to implement a sound contradiction removal algorithm based on the Well Founded Semantics meta-interpreters of [19, 18], so as to obtain three-valued revisions (to the undefined truth-value) of (negative) assumptions. To obtain a two-valued revision, assumptions are changed instead into their complements. Since this may introduce fresh contradictions, the contradiction removal algorithm must be iterated. So the algorithm consists of iterated two-valued partial revisions as directed by three-valued revision oppurtunities.

As a result we obtain more accumulating evidence that a large class of problems can be solved with a contradiction removal approach. Its relationship to abduction is studied in [1, 14]. In short, minimal contradiction removal is comparable to maximal consistent abduction.

[3] unifies the abductive and consistency-based approaches to diagnosis, and so, for generality, we present a methodology that transforms a diagnostic problem of [3] into an extended logic program and solve it with contradiction removal. Another unifying approach to diagnosis with logic programming [23] uses Generalised Stable Models [11]. They present criticisms of Console and Torasso's approach which do not carry over to our representation, ours having the advantage of a more expressive language: explicit negation as

well as negation as failure.

We also set forth a method to debug pure Prolog programs, showing that declarative debugging [12] can be envisaged as contradiction removal, and providing a simple and clear solution to this problem. Furthermore we show how diagnostic problems can be solved with contradiction removal applied to the artifact's representation in logic plus observations. Declarative debugging can be used to diagnose blueprint specifications of artifacts.

Section 2 describes the language and notation adopted. Section 3 recaps $CRSX$ and present the new two-valued contradiction removal definitions and algorithms. Sections 4 and 5 apply these procedures to diagnostic problems and to declarative debugging. All examples and algorithms were implemented and successfully tested using a Prolog meta-interpreter.

2 Language

Given a first order language $Lang$, an extended logic program is a set of rules and integrity rules of the form

$$H \leftarrow B_1, \ldots, B_n, not\, C_1, \ldots, not\, C_m \quad (m \geq 0, n \geq 0)$$

where $H, B_1, \ldots, B_n, C_1, \ldots, C_m$ are objective literals, and in integrity rules H is $\perp$ (contradiction). When $n = m = 0$, H is an alternative representation for rule $H \leftarrow$. An objective literal is either an atom A or its explicit negation $\neg A$, where $\neg\neg A = A$. $not\, L$ is called a default or negative literal. Literals are either objective or default ones. The default complement of objective literal L is $not\, L$, and of default literal $not\, L$ is L. A rule stands for all its ground instances wrt $Lang$. The notation $H \leftarrow \mathcal{B}$ is also used to represent a rule, where set $\mathcal{B}$ contains the literals in its body. For every pair of objective literals $\{L, \neg L\}$ in $Lang$ we implicitly assume the rule $\perp \leftarrow L, \neg L$.

As in [PP90], we expand our language by adding to it the proposition $\mathbf{u}$ such that for every interpretation I, $I(\mathbf{u}) = \frac{1}{2}$. By a non-negative program we mean a program whose premises are either objective literals or $\mathbf{u}$. Given a program P we denote by $\mathcal{H}_\mathcal{P}$ (or simply $\mathcal{H}$) its Herbrand base. If S is a set of literals $\{L_1, \ldots, L_n\}$, by $not\, S$ we mean the set $\{not\, L \mid L \in S\}$.

If S is a set of literals then we say S is *contradictory* iff there is objective literal L such that $\{L, \neg L\} \subseteq S$. S is *contradictory* wrt to L.

3 Revising Contradictory Extended Logic Programs

Once we introduce explicit negation programs are liable to be contradictory. Next we review the Contradiciton Removal Semantics ($CRSX$) of [16].

Example 3.1 Consider $P = \{a; \neg a \leftarrow not\, b\}$. Since we have no rules for b, by CWA it is natural to accept $not\, b$ as true. By the second rule in P wehave $\neg a$, leading to an inconsistency with the fact a.

It is arguable that the CWA may not be held of atom b since it leads to contradiction. Revising such $CWAs$ is the basis of our contradiction removal methods. We show below two different ways of how to avoid this form of contradiction, by preventing the incorrect CWA on b. Three questions that must be answered to select a particular contradiction removal process:

1. For which literals is revision of their truth-value allowed ?
2. To what truth values do we change the revisable literals ?
3. How to choose among possible revisions ?

The options taken here are clarified in the discussion in sections 3.2 and 3.3, giving two different answers to these questions. Both use the same criteria to answer 1 and 3, but differ on the second one. The first way of removing contradiction gives $\{a, not\ \neg a, not\ \neg b\}$ as the intended meaning of P, where b is revised to *undefined*. The second gives $\{a, b, not\ \neg a, not\ \neg b\}$, by revising b to true.

3.1 Contradictory Well Founded Model

To revise contradictions we need to identify the contradictory sets of consequences 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. Furthermore, the coherence principle is enforced at each step. It states that, for any objective literal, if $\neg L$ is entailed by the semantics then $not\ L$ is too:

$$\neg L \ \Rightarrow not\ L \quad (CP)$$

The following example provides an intuitive preview of what we intend:

Example 3.2 Consider program P:

$$\text{a}\leftarrow \text{not b.}\ \text{(i)} \qquad \neg\text{a}\leftarrow\text{not c.}\ \text{(ii)} \qquad \text{d}\leftarrow\text{a.}\ \text{(iii)} \qquad \text{e}\leftarrow\neg\text{a.}\ \text{(iv)}$$

1. $not\ b$ and $not\ c$ 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. $not\ a$ and $not\ \neg a$ hold from 2 and inference rule (CP).
4. d and e hold from 2 and rules (iii) and (iv).
5. $not\ d$ and $not\ e$ hold from 3 and rules (iii) and (iv), as they are the only rules for d and e.
6. $not\ \neg d$ and $not\ \neg e$ hold from 4 and inference rule (CP).

The whole set of consequences is

$$\{\neg a, a, not\ a, not\ \neg a, not\ b, not\ c, d, not\ d, not\ \neg d, e, not\ e, not\ \neg e\}$$

Definition 3.1 (Pseudo-interpretation) *A pseudo-interpretation, or p-interpretation, is a possibly contradictory set of ground literals from Lang.*

One can define a pseudo Well Founded model, as the F-least fixed point of a new operator, extending the Θ operator of [24] to p-interpretations [16].

3.2 Contradiction Removal Sets

To revise contradiction the first issue to consider is which default literals true by CWA are allowed to change their truth values. We simplify the approach of [16] along the lines of [15] taking as candidates for change default literals true by CWA in the pseudo Well Founded Model. By making this simplification we can give a syntactic condition for electing the revisable literals, in contradistinction to the semantic one of [16].

Definition 3.2 (Revisables) *The revisables of a program P are the elements of a chosen subset of $Rev(P)$, the set of all default literals $not\, L$ having no rules for L in P, and so true by CWA.*

Next we identify the revisables supporting contradiction. Their revision to *undefined* can remove contradiction, by withdrawing the support of $CWAs$ on which it rests, and doesn't introduce new contradictions. But first we define support of a literal in general; intuitively, a support of a literal consists of the literals in nodes of a derivation for it in the pseudo WFM:

Definition 3.3 (Support set of a literal) *A support set of a literal of the (pseudo) Well Founded Model M_P of a program P, denoted by $SS(L)$, is obtained as follows:*

1. *If L is an objective literal in M_P then for each rule $L \leftarrow \mathcal{B}$ in P, such that $\mathcal{B} \subseteq M_P$ there is one $SS(L)$ formed by the union of $\{L\}$ with one SS for each $B_i \in \mathcal{B}$.*
2. *If L is a default literal $not\, A \in M_P$:*
   - (a) *if no rules exist for A in P then a support set of L is $\{not\, A\}$.*
   - (b) *if rules for A exist in P then choose from each rule with non-empty body a single literal such that its default complement belongs to M_P. For each such multiple choice there is one SS for $not\, A$ formed by the union of $\{not\, A\}$ with one SS of each default complement of the chosen literals.*
   - (c) *if $\neg A$ belongs to M_P then there exist,* additionally, *support sets SS of $not\, A$ equal to each $SS(\neg A)$.*

Example 3.3 The pseudo Well Founded Model M_P of:

¬p←not c.	p←t.	b←c, a.	¬b←not e.	a.
	p←a,not b.	b←d.		

is $\{a, not\, \neg a, not\, b, \neg b, not\, c, not\, \neg c, not\, d, not\, \neg d, not\, e, not\, \neg e, not\, t, not\, \neg t, p, \neg p, not\, p, not\, \neg p, \bot\}$. There are two support sets for $not\, b$:

$$SS_1(not\, b) = \{not\, b\} \cup SS(not\, c) \cup SS(not\, d) \qquad \textit{by rule 2b}$$
$$SS_1(not\, b) = \{not\, b\} \cup \{not\, c\} \cup \{not\, d\} = \{not\, b, not\, c, not\, d\} \qquad \textit{by rule 2a}$$

Notice that the other possibility of choosing literals for $SS(not\, b)$, i.e. $SS_1(not\, b) = \{not\, b\} \cup SS(not\, a) \cup SS(not\, d)$, can't be considered because $not\, a$ doesn't belong to M_P. The other support set for $not\, b$ is obtained using rule 2c:

$$\begin{array}{ll} SS_2(not\, b) = SS(\neg b) & \textit{by rule } 2c \\ SS_2(not\, b) = \{\neg b\} \cup SS(not\, e) & \textit{by rule } 1 \\ SS_2(not\, b) = \{\neg b, not\, e\} & \textit{by rule } 2a \end{array}$$

Now the support sets for the objective literal p are easily computed:

$$\begin{array}{ll} SS(p) = \{p\} \cup SS(a) \cup SS(not\, b) & \textit{by rule } 1 \\ SS(p) = \{p\} \cup \{\} \cup SS(not\, b) & \textit{by rule } 1 \textit{ (the only rule for a is fact a)} \end{array}$$

So $SS_1(p) = \{p\} \cup SS1(not\, b) = \{p, not\, b, not\, c, not\, d\}$ and $SS_2(p) = \{p\} \cup SS2(not\, b) = \{p, \neg b, not\, e\}$. $\neg p$ has the unique support set $\{\neg p, not\, c\}$.

Proposition 3.1 (Existence of support sets) *Every literal belonging to the pseudo WFM of a program P has at least one support set $SS(L)$.*

Next we must find on which revisable literals contradiction rests, by finding the revisables belonging to the supports of $\perp$. Formally:

Definition 3.4 (Assumption set of a literal wrt revisables)
Given revisables R of program P, an assumption set of L wrt R is the set $AS(L, R) = SS(L) \cap R$, where $SS(L)$ is a support set of L.

Example 3.3 (cont.) Let $R = Rev(P) = \{not\, \neg a, not\, c, not\, \neg c, not\, d, not\, \neg d, not\, e, not\, \neg e, not\, t, not\, \neg t\}$. The assumption sets of p wrt R are $AS1(p) = SS1(p) \cap R = \{not\, c, not\, d\}$ and $AS2(p) = SS2(p) \cap R = \{not\, e\}$. The only assumption set wrt R of $\neg p$ is $AS1(\neg p) = SS1(\neg p) \cap R = \{not\, c\}$.

We define a spectrum of possible revisions using the notion of hitting set:

Definition 3.5 (Hitting set) *A hitting set of a collection C of sets is formed by the union of one non-empty subset from each $S \in C$. A hitting set is minimal iff no proper subset is a hitting set. If $\{\} \in C$, C has no hitting sets.*

Definition 3.6 (Removal set) *A removal set of a literal L of a program P wrt revisables set R is a hitting set of all assumption sets $AS(L, R)$.*

We can revise contradictory programs by undefining the literals of a removal set of $\perp$ (a removal set literal $not\, L$ is undefined in P by adding to it the inhibition rule $L \leftarrow not\, L$). This defines the possible revisions of a contradictory program. We answer the third question by prefering to undefine minimal sets of revisables:

Definition 3.7 (Contradiction removal set) *A contradiction removal set of P wrt revisables R is a a minimal removal set of $\bot$ wrt R.*

Example 3.3 (cont.) The assumption sets of $\bot$ wrt R are $\{not\ c, not\ d\}$ and $\{not\ c, not\ e\}$. The removal sets are (RS_1 and RS_4 being minimal):

$$\begin{array}{ll} RS_1(\bot, R) = \{not\ c\} & RS_4(\bot, R) = \{not\ d, not\ e\} \\ RS_2(\bot, R) = \{not\ c, not\ e\} & RS_5(\bot, R) = \{not\ c, not\ d, not\ e\} \\ RS_3(\bot, R) = \{not\ c, not\ d\} & \end{array}$$

A program is not revisable if $\bot$ has a support set without revisable literals.

Definition 3.8 (Revisable program) *A program is revisable wrt a set of revisables R iff it has contradiction removal sets wrt R.*

The $CRSs$ are minimal hitting sets of the collection of assumptions sets of $\bot$. [25] gives an "algorithm" for computing minimal diagnosis, called DIAGNOSE (with a bug detected and corrected in [10]). It is known [7] that this problem is NP-complete. DIAGNOSE can be used to compute $CRSs$, needing only the definition of the function Tp refered there. Our Tp can be easily built from a top-down derivation procedure for the pseudo Well Founded Model adapted from [19]. It is presented in an extended version of this paper.

3.3 Contradiction Removal with Two-valued Assumptions

In this section we perform contradiction removal by adding to the original program default complements of revisables (instead of undefining them by adding inhibition rules), thereby revising default assumptions from true to false. For simplicity we assume $R = Rev(P)$.

Definition 3.9 (Set of assumptions of a program) *A set A of objective literals is a set of assumptions of program P iff $\forall L \in A \Rightarrow not\ L \in Rev(P)$.*

Definition 3.10 (Submodel of a prog. wrt a set of assumptions) *Let A be a set of assumptions of P. The Submodel of P wrt A, $SubM(A)$, is the (possibly contradictory) pseudo Well Founded Model of $P \cup A$.*

Definition 3.11 (Set of revising assumptions of a program) *A set of assumptions A of P is a set of revising assumptions iff $\bot \notin SubM(A)$; A is a 2-valued revision of P. Otherwise A is a set of non-revising assumptions.*

Adding the default complement of $CRSs$ as of positive assumptions to a logic program may lead to new contradictions. The revision process must be iterated. The power of this form of contradiction removal rests on this feature.

3.4 Computing Minimal Revising Assumptions

Now we present an iterative algorithm to compute the minimal sets of revising assumptions of a program P wrt a set of revisables R, which is sound and complete for the finite case. Intuitively, this algorithm rests on a repeated application of the algorithm to compute the $CRSs$ of the original program (assuming the original program is revisable, otherwise the algorithm stops after the first step). To each CRS there corresponds a set of revised assumptions obtained by taking the default complement of their elements. The algorithm then adds, non-deterministically, one at a time, each of these sets of assumptions to the original program. One of three cases occurs: (1) the program thus obtained is non-contradictory and we are in the presence of a possibly minimal revising set of assumptions; (2) the new program is contradictory and non-revisable (and this fact is recorded by the algorithm to prune out other contradictory programs obtained by it); (3) the new program is contradictory but revisable and this very same algorithm is iterated until we finitely attain one of the two other cases. In the end, the minimal revising sets of assumptions obtained can be used to revise the original program to non-contradictory ones.

Algorithm 3.1 (Minimal revising assumptions of a program)

Input: *A logic program P, possibly contradictory, and a set R of revisables.*
Output: *The sets of minimal revising assumptions of P wrt R (in AS_i).*

$AS_0 := \{\{\}\}$; $Cs := \{\}$; $i := 0$
repeat $AS_{i+1} := \{\}$;
    for each $A \in AS_i$
        if $\neg\exists C \in Cs : C \subseteq A$
            if $Rev(P, A) \models \perp$
                if $Rev(P, A)$ *is revisable*
                    for each $CRS_j(R)$ *of* $P \cup A$
                        Let $NAs := A \cup not\, CRS_j(R)$;
                        $AS_{i+1} := AS_{i+1} \cup \{NAs\}$
                    endfor
                else $Cs := Cs \cup \{A\} - \{A' \in Cs : A \subseteq A'\}$
                endif
            else $AS_{i+1} := AS_{i+1} \cup \{A\}$
            endif
        endif
    endfor
    $ASi + 1 := MinimalSetsOf(AS_{i+1})$; $i := i + 1$
until $AS_i = AS_{i-1}$.

This algorithm can terminate after executing only one step ($i = 1$) when the program is either non-contradictory or contradictory and non-revisable.

Example 3.4 Detailed execution for contradictory program P:

$$p \leftarrow \text{not } a. \qquad \neg p \leftarrow \text{not } c. \qquad x. \qquad \neg x \leftarrow c, \text{not } a, \text{not } b.$$

with set of revisables $R = \{not\ a, not\ \neg a, not\ b, not\ \neg b, not\ c, not\ \neg c\}$.

- $i = 0 : AS_0 = \{\{\}\}, Cs = \{\}$.

  The only A in AS_0 is $\{\}$. As $\perp \in SubM(\{\})$, with $CRS_1 = \{not\ a\}$ and $CRS_2 = \{not\ c\}$, $AS_1 = \{\{a\}, \{c\}\}$.

- $i = 1 : AS_1 = \{\{a\}, \{c\}\}, Cs = \{\}$.

  For $A = \{a\}$, $SubM(\{a\})$ is non-contradictory. The other option is $A = \{c\}$, with $\perp \in SubM(\{c\})$, so $CRS_1 = \{not\ a\}$ and $CRS_2 = \{not\ b\}$. Thus $AS_2 = \{\{a\}, \{b, c\}\}$ since $\{a\}$ is in $\{a, c\}$.

- $i = 2 : AS_1 = \{\{a\}, \{b, c\}\}, Cs = \{\}$.

  With $A = \{a\}$ and $A = \{b, c\}$ $SubM(A)$ is non-contradictory, which implies $AS_3 = AS_2$ and so the algorithm stops.

The sets of minimal revising assumptions for this program wrt to R are $A_1 = \{a\}$ and $A_2 = \{b, c\}$. Note the need for retaining only minimal sets of assumptions to get the desired result without making useless computation.

Theorem 3.1 (Soundness) *If algorithm 3.1 terminates in iteration i, AS_i is the collection of all sets of minimal revising assumptions of P wrt R.*

Theorem 3.2 (Completeness) *For finite R algorithm 3.1 stops.*

This contradiction removal process is very similar to abduction and it can be shown that algorithm 3.1 is NP-complete like other abductive procedures [2, 6, 26].

4 Application to Declarative Debugging

We can apply contradiction removal to perform debugging of terminating pure Horn Prolog programs, assuming a program stands for its ground version. In [21] we generalize to normal programs.

Besides looping there are only two other kinds of error [12]: wrong solutions and finitely missing solutions.

4.1 Debugging Wrong Solutions

Consider the buggy program P, where $a(2)$ succeeds wrongly:

a(1).	b(2).	c(1,X).
a(X)←b(X),c(Y,Y).	b(3).	c(2,2).

What are the minimal causes of this bug? There are three: the second rule for a has a bug; $b(2)$ should not hold in P; or neither $c(1, X)$ nor $c(2, 2)$ should hold in P.

This type of error (and its causes) is easily detected using contradiction removal by means of a simple transformation applied to the original program:

- Add default literal $not\ ab_i([X_1, X_2, \ldots, X_n])$ to the body of each i-th rule of P, where n is its arity and $X_1, X_2, \ldots, X_n$ its head arguments.

Applying this program transformation to P we get the new program P_1:

a(1) $\leftarrow$not ab$_1$([1]).	a(X) $\leftarrow$b(X), c(Y,Y), not ab$_2$([X]).
b(2) $\leftarrow$not ab$_3$([2]).	b(3) $\leftarrow$not ab$_4$([3]).
c(1,X)$\leftarrow$not ab$_5$([1,X]).	c(2,2)$\leftarrow$not ab$_6$([2,2]).

If $p(X_1, X_2, \ldots, X_n)$ succeeds wrongly in P add to P_1 fact $\neg p(X_1, X_2, \ldots, X_n)$, and revise P_1 to find the minimal possible causes of the wrong solution, using as revisables all $not\ ab_i/1$ literals.

Example 4.1 $a(2)$ wrongly succeeds in P; adding $\neg a(2)$ to P_1 we get minimal revisions $\{ab_1([2])\}$, $\{ab_3([2])\}$ and $\{ab_5([1, 1]), ab_5([2, 2])\}$ as expected.

To automatically filter revisions invoking correct clauses insert rules in the transformed program of the form:

- $\neg ab_i([X_1, X_2, \ldots, X_n]) \leftarrow valid(p(X_1, X_2, \ldots, X_n)), valid(B_1), \ldots, valid(B_k)$ for the i-th rule $p(X_1, X_2, \ldots, X_n) \leftarrow B_1, \ldots, B_k$ of P.

Selection among revisions is achieved by asserting facts of the form $valid(A)$ in the transformed program, stating A to be a valid solution in P.

Example 4.1 (cont.) The new program transformation will add to P_1:

¬ab$_1$([1]) $\leftarrow$valid(a(1)).	¬ab$_2$([X]) $\leftarrow$valid(a(X)),valid(b(X)),valid(c(Y,Y)).
¬ab$_3$([2]) $\leftarrow$valid(b(2)).	¬ab$_4$([3]) $\leftarrow$valid(b(3)).
¬ab$_5$([1,X])$\leftarrow$valid(c(1,X)).	¬ab$_6$([2,2])$\leftarrow$valid(c(2,2)).

Suppose that you are sure the first rule for $c/2$ is correct and also the rules for b. So you add $\{valid(c(1, X)), valid(b(2)), valid(b(3))\}$ to P_1 and get the only minimal revision $\{ab_2([2])\}$.

It is also possible to explicitly state some rule is correct by adding $\neg ab_i(\_)$.

4.2 Debugging Missing Solutions

Suppose now a program should succed on some goal but finitely fail. This is the missing solution problem. Say, for instance, $a(4)$ should succeed in program P above. Which are the minimal sets of facts that added to P make $a(4)$ succeed ? $a(4)$ or $b(4)$. Such sufficient solutions identify uncovered goals.

To find this type of bug it suffices to add for each predicate p with arity n the rule to P_1:

- $p(X_1, X_2, \ldots, X_n) \leftarrow missing(p(X_1, X_2, \ldots, X_n))$.

Then all that's needed to state q has missing solution $q(X_1, X_2, \ldots, X_n)$ is to add to P_1 the integrity rule $\perp \leftarrow not\, q(X_1, X_2, \ldots, X_n)$. Then a contradiction arises, and P_1 is revised, using as revisables $not\, missing(A)$, for all atoms A.

Example 4.2 The transformed program P_1 is P plus the rules:

a(X)←missing(a(X)).   b(X)←missing(b(X)).   c(X,Y)←missing(c(X,Y)).

To find the possible causes of the missing solution to $a(4)$, add integrity rule $\perp \leftarrow not\, a(4)$ and obtain, as expected, the two minimal revisions $\{missing(a(4))\}$ and $\{missing(b(4))\}$.

The filtering of revisions can be done by asserting facts $\neg missing(X)$, to the effect that X is not a missing solution.

Finally, the two program transformations can be applied simultaneously in order to achieve the detection and correction of both types of errors. The two sets of revisables can be conjoined without problem, as the two types of error don't interfere in Horn programs. The debugging of normal programs [21] makes simultaneous use of both, in a straightforward way.

5 Application to Diagnosis

In this section we describe a general program transformation that translates diagnostic problems (**DP**), in the sense of [3], into logic programs with integrity rules. By revising this program we obtain the diagnostic problem's minimal solutions, i.e. the diagnoses. The unifying approach of abductive and consistency-based diagnosis presented by these authors enables us to represent easily and solve a major class of diagnostic problems using two-valued contradiction removal. Similar work has been done by [23] using Generalised Stable Models [11].

We start by making a short description of a diagnostic problem as defined in [3, 5]. A **DP** is a triple consisting of a system description, inputs and observations. The system is modelled by a Horn theory describing the devices, their behaviours and relationships. In this diagnosis setting, each component of the system to be diagnosed has a description of its possible behaviours with the additional restriction that a given device can only be in a single mode of a set of possible ones. There is a mandatory mode in each component modelled, the correct mode, that describes correct device behaviour; the other mutually exclusive behaviour modes represent possible faulty behaviours.

Having this static model of the system we can submit to it a given set of inputs (contextual data) and compare the results obtained with the observations predicted by our conceptualized model. Following [3] the contextual data and observation part of the diagnostic problem are sets of parameters of the form parameter(value) with the restriction that a given parameter can only have one observed valued.

From these introductory definitions [3] present a general diagnosis framework unifying the consistency-based and abductive approaches. These authors translate the diagnostic problem into abduction problems where the abducibles are the behaviour modes of the various system components. From the observations of the system two sets are constructed: Ψ^+, the subset of the observations that must be explained, and $\Psi^- = \{\neg f(X) : f(Y)$ is an observation, for each admissible value X of parameter f other than $Y\}$. A diagnosis is a minimal consistent set of abnormality hypotheses, with additional assumptions of correct behaviour of the other devices, that consistently explain some of the observed outputs: the program plus the hypotheses must derive (cover) all the observations in Ψ^+ consistent with Ψ^-. By varying the set Ψ^+ a spectrum of different types of diagnosis is obtained.

We show that it is always possible to compute the minimal solutions of a diagnostic problem by computing the minimal revising assumptions of a simple program transformation of the system model.

Example 5.1 Consider the following partial model of an engine, with only one component *oil_cup*, which has behaviour modes *correct* and *holed* [3]:

oil_below_car(present)	←holed(oil_cup).
oil_level(low)	←holed(oil_cup).
oil_level(normal)	←correct(oil_cup).
engine_temperature(high)	←oil_level(low), engine(on).
engine_temperature(normal)	←oil_level(normal), engine(on).

An observation is made of the system, and it is known that the engine is on and that there is oil below the car. The authors study two abduction problems corresponding to this **DP** :

1. $\Psi^+ = \{oil\_below\_car(present)\}$ and $\Psi^- = \{\}$ (Poole's view of a diagnostic problem [22]) with minimal solution $W_1 = \{holed(oil\_cup)\}$.
2. $\Psi^+ = \Psi^- = \{\}$ (De Kleer's **DP** view [4]) with minimal solution $W_2 = \{\}$.

To solve abduction problem 1 it is necessary to add the following rules:

$$\begin{array}{ll}\bot & \leftarrow \text{not oil\_below\_car(present).} \\ \text{correct(oil\_cup)} & \leftarrow \text{not ab(oil\_cup).} \\ \text{holed(oil\_cup)} & \leftarrow \text{ab(oil\_cup),fault\_mode(oil\_cup, holed).}\end{array}$$

The above program has only one minimal revision $\{ab(oil\_cup), fault\_mode(oil\_cup, holed)\}$ as wanted.

To solve the second problem, the transformed program has the same rules of the program for problem P, except the integrity constraint–it is not necessary to cover any set of obervations. The program thus obtained is non-contradictory having minimal revision $\{\}$.

Next, we present the general program transformation which turns a diagnostic abduction problem into a contradiction removal problem.

Theorem 5.1 *Given an abduction problem (**AP**) corresponding to a diagnostic problem, the minimal solutions of **AP** are the minimal revising assumptions of the modelling program plus contextual data and the following rules:*

1. $\bot \leftarrow not\ obs(v)$*, for each* $obs(v) \in \Psi^+$.
2. $\neg obs(v)$*, for each* $obs(v) \in \Psi^-$.

and for each component c_i *with distinct abnormality behaviour modes* b_j *and* b_k*:*

3. $correct(c_i) \leftarrow not\ ab(c_i)$.
4. $b_j(ci) \leftarrow ab(c_i), fault\_mode(c_i, b_j)$.
5. $\bot \leftarrow fault\_mode(c_i, b_j), fault\_mode(c_i, b_k)$ *for each* b_j, b_k.

with revisables $fault\_mode(c_i, b_j)$ *and* $ab(c_i)$.

We don't give a detailed proof of this result but take into consideration:

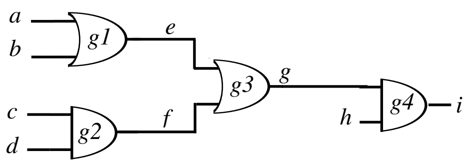
- Rule 1 ensures that, for each consistent set of assumptions, $obs(v) \in \Psi^+$ must be entailed by the program.
- Rule 2 guarantees the consistency of the sets of assumptions with Ψ^-.
- Rules 4 and 5 deal and generate all the possible mutually exclusive behaviours of a given component.

Finally, in no revision there appears the literal $fault\_mode(c, correct)$, thus guaranteeing that minimal revising assumptions are indeed minimal solutions to the **DP**.

The concept of declarative debugging, see section 4, can be used to aid in the development of logic programs and in particular to help the construction of behavioural models of devices. Firstly, a Prolog prototype or blueprint of the component is written and debugged using the methodology presented in that section. After the system is constructed, the diagnostic problems can be solved using contradiction removal as described above, in the correct blueprint.

Now we present an extended example of a classical, circuit diagnosis, and show its solution using our program transformation:

Example 5.2 Consider the circuit:



with inputs $a = 0$, $b = 1$, $c = 1$, $d = 1$, $h = 1$ and (incorrect) output 0. Its behavioural model is:

% Normal behaviour of and gates	% Faulty behaviour
and_gate(G,I1,I2,1)←correct(G).	and_gate(G,1,1,0)←abnormal(G).
and_gate(G,0,1,0) ←correct(G).	and_gate(G,0,1,1)←abnormal(G).
and_gate(G,1,0,0) ←correct(G).	and_gate(G,1,0,1)←abnormal(G).
and_gate(G,0,0,0) ←correct(G).	and_gate(G,0,0,1)←abnormal(G).

And a similar set of rules for *or* gates. According to the program transformation two auxiliary rules are needed:

correct(G)←not ab(G). abnormal(G)←ab(G).

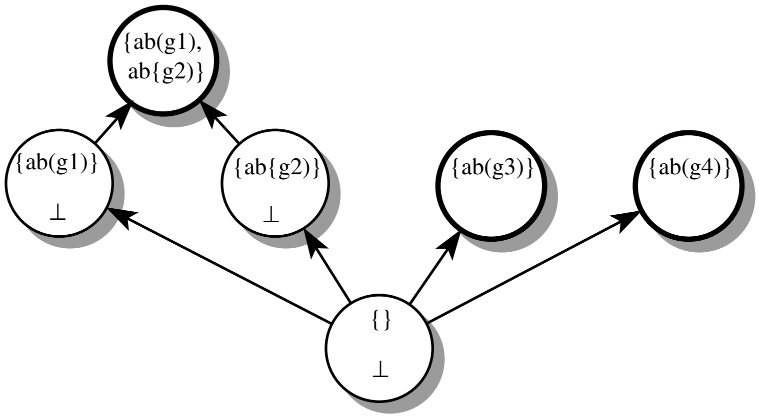
and the description of the circuit and its connections:

% Nodes
node(a,0). node(b,1). node(c,1). node(d,1). node(h,1).

% Connections
node(e, E) ←node(a, A), node(b, B), or_gate(g1, A, B, E).
node(f, F) ←node(c, C), node(d, D), and_gate(g2, C, D, F).
node(g, G)←node(e, E), node(f, F), or_gate(g3, E, F, G).
node(i, I) ←node(g, G), node(h, H), and_gate(g4, G, H, I).

Value consistency: ¬node(i,0)←node(i,1). ¬node(i,1)←node(i,0).
Observed output of the circuit: node(i, 0).

The minimal solutions to this problem are highlighted in the next figure. As expected, the minimal revising assumptions $\{ab(g1), ab(g2)\}$, $ab(g3)$ and $\{ab(g4)\}$ are the minimal solutions to the diagnosis problem.



Acknowledgements

We thank JNICT Portugal and ESPRIT project Compulog 2 for their support.

References

[1] J. J. Alferes and L. M. Pereira. Contradiction: when avoidance equal removal. Part I. In *4th Int. Ws on Extensions of Logic Programming*. University of St. Andrews, 1993.

[2] T. Bylander, D. Allemang, M. C. Tanner, and J. R. Josephson. Some results concerning the computational complexity of abduction. In *Proc. KR-89*, pages 44–45. Morgan Kaufmann, 1989.

[3] L. Console and P. Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, 7:133–141, 1991.

[4] J. de Kleer and B.C. Williams. Diagnosing multiple faults. *AI*, 32:97–130, 1987.

[5] J. de Kleer and B.C. Williams. Diagnosis with behavioral modes. In *Proc. IJCAI'89*, pages 1329–1330, 1989.

[6] G. Friedrich, G. Gottlob, and W. Nejdl. Physical impossibility instead of fault models. In *Proc. AAAI-90*, pages 331–336, 1990.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, 1979.

[8] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In D. Warren and P. Szeredi, editors, *7th ICLP*, pages 579–597. MIT Press, 1990.

[9] M. Gelfond and V. Lifschitz. Representing actions in extended logic programming. In K. Apt, editor, *Proc. IJCSLP'92*. MIT Press, 1992.

[10] R. Greiner, B. A. Smith, and R. W. Wilkerson. A correction to the algorithm in reiter's theory of diagnosis. *AI*, 41:79–88, 1989.

[11] A. C. Kakas and P. Mancarella. Generalised stable models: A semantics for abduction. In *Proc. ECAI'90*, pages 401–405, 1990.

[12] J. W. Lloyd. Declarative error diagnosis. *New Generation Computing*, 5(2):133–154, 1987.

[13] L. M. Pereira and J. J. Alferes. Well founded semantics for logic programs with explicit negation. In B. Neumann, editor, *Proc. ECAI'92*, pages 102–106. John Wiley, 1992.

[14] L. M. Pereira and J. J. Alferes. Contradiction: when avoidance equal removal. Part II. In *4th Int. Ws on Extensions of Logic Programming*. University of St. Andrews, 1993.

[15] L. M. Pereira, J. J. Alferes, and J.N. Aparício. Contradiction removal within well founded semantics. In et al. A. Nerode, editor, *Proc. Logic Programming and Non-Monotonic Reasoning'91*, pages 105–119. MIT Press, 1991.

[16] L. M. Pereira, J. J. Alferes, and J.N. Aparício. Contradiction removal semantics with explicit negation. In *Proc. Applied Logic Conf.*, Amsterdam, 1992. ILLC.

[17] L. M. Pereira, J. J. Alferes, and J.N. Aparício. Logic programming for nonmonotonic reasoning. In *Proc. Applied Logic Conf.*, Amsterdam, 1992. ILLC.

[18] L. M. Pereira, J. J. Alferes, and C. Damásio. The sidetracking principle applied to well founded semantics. In *Proc. Simpósio Brasileiro de Inteligência Artificial SBIA'92*, pages 229–242, 1992.

[19] L. M. Pereira, J.N. Aparício, and J. J. Alferes. Derivation procedures for extended stable models. In *Proc. IJCAI-91*. Morgan Kaufmann, 1991.

[20] L. M. Pereira, J.N. Aparício, and J. J. Alferes. Nonmonotonic reasoning with well founded semantics. In K. Furukawa, editor, *Proc. ICLP'91*, pages 475–489. MIT Press, 1991.

[21] L. M. Pereira, C. V. Damásio, and J. J. Alferes. Debugging by diagnosing assumptions. Technical report, Univ. Nova de Lisboa, 1993. Submitted to AADEBUG'93.

[22] D. Poole. Normality and faults in logic-based diagnosis. In *Proc. IJCAI-89*, pages 1304–1310, 1989.

[23] C. Preist and K. Eshghi. Consistency-based and abductive diagnoses as generalised stable models. In *Proc. FGCS'92*. ICOT, Omsha 1992.

[24] H. Przymusinska and T. Przymusinski. *Semantic Issues in Deductive Databases and Logic Programs*, pages 321–367. Formal Techniques in Artificial Intelligence. A sourcebook. Elsevier, 1990.

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

[26] B. Selman and H. J. Levesque. Abductive and default reasoning: A computational core. In *Proc. AAAI-90*, pages 343–348, 1990.