Diagnosing Dynamic Systems in A-Prolog

Michael Gelfond
Computer Science Dept.
Texas Tech University
mgelfond@cs.ttu.edu

Joel Galloway
Computer Science Dept.
University of Texas at El Paso
jgallo@cs.utep.edu

Introduction

In the first part of this paper we address the problem of diagnosing faulty behavior of a dynamic system. Similar to the work in (Baral, McIlraith & Son 2000) our approach is based on the semantics of action languages (Gelfond & Lifschitz 1998). Unlike (Baral, McIlraith & Son 2000) we do not use action language $\mathcal{L}$. Instead we assume that the set of all possible trajectories of the system is described by a *system description* of an action description language $\mathcal{AL}_d$ (Baral & Gelfond 2000) (possibly augmented by logic programming rules of A-Prolog). We believe that this language is sufficiently powerful for a wide range of applications. The use of $\mathcal{AL}_d$ allows us to give a definition of diagnosis which, we believe, is simpler than related definitions of (Baral, McIlraith & Son 2000). Simplicity of the definition and the recent discoveries of the close relationship between A-Prolog and reasoning about effects of actions allow us to develop a collection of simple algorithms for computing diagnoses. The algorithms are based on the ideas from answer set programming (Marek & Truszczynski 1999; Niemela 1999; Lifschitz 1999) and are implemented on top of SMODELS (Niemela & Simons 1996) - a system for computing stable models of logic programs. According to our definition all diagnoses are created equal. In practice however, some diagnoses are more equal than others. Selection of the "best" diagnosis is often based on some heuristic information about the plausibility of different types of faults. By way of example we show how our choice of A-Prolog as a knowledge representation language allows us to incorporate this heuristic information in the corresponding diagnostic systems. The heuristics improve the quality of diagnoses as well as efficiency of computation.

The second part of the paper describes a software architecture for a **DPS** *(diagnostic problem solver)* - an agent capable of discovering and diagnosing faulty behavior of the system, testing it's suspected faulty components, and repairing components which indeed are found to be faulty. The architecture, of course, strongly relies on the ideas developed in the first section.

Background

Let S be a physical system consisting of components $c_1, \dots, c_n$ and $\Sigma = \{A, F\}$. Elements of A denote (elementary) actions capable of changing S's states; F is a collection of fluents - propositions whose truth values depend on time. By fluent literals we mean fluents and their negations. A state of S can be characterized by a complete and consistent set of fluent literals. We assume that A consists of a set A_s of actions performed by the system (or an agent controlling the system) and a set A_e of exogenous actions whose occurrence can cause system components to malfunction. S is fully characterized by a system description SD consisting of rules of A-Prolog defining components of S and their relationships, fluents and actions of Σ, dynamic and static causal laws determining the effects of actions, and the actions' executability conditions. We assume that SD has a unique answer set which defines an *action description*, D, in *action description language* $\mathcal{AL}_d$. Semantics of $\mathcal{AL}_d$ associates D with a transition diagram T_d representing all possible trajectories of S. States of T_d are states of S and arcs are labeled by *compound actions* - sets of elements from A. In our further discussion we abuse the notation and identify SD with D and action $\{a\}$ with a. Causal laws of D can be divided into two parts. The first part, D_n, contains laws describing normal behavior of the system. Their bodies usually contain special fluent literals of the form $\neg ab(c)$. The statement $ab(c)$ is read as "component c of system S is abnormal". Its use in diagnosis goes back to (Reiter 1987). The second part, D_b, describes effects of exogenous actions damaging the components. Such laws normally contain relation ab in the head or positive parts of the bodies. The existence of the second component (absent in (Reiter 1987)), will be essential for our approach to computing diagnosis.

History of S is specified by a collection $\Gamma(t_c)$ of statements in the *history description language* $\mathcal{AL}_h$. We assume that the system's time is discrete, t_c is the current moment of time and the statements of $\Gamma(t_c)$ have the

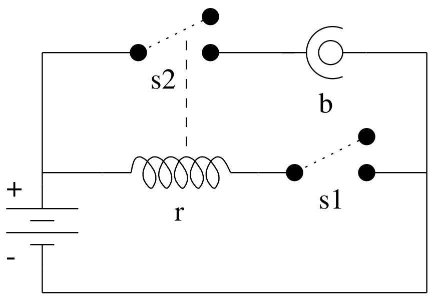

Figure 1: $\mathcal{AC}$

form:

$obs(l,t)$ (fluent literal l was observed to be true at moment t), and

$hpd(a,t)$ (elementary action $a \in A$ was observed to happen at moment t) where $t < t_c$.

Let D be an action description and Γ be a history of the system up to a moment n. We say that a path $\sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n$ in T_d is a *model* of Γ w.r.t. D iff

1. $a_k = \{a : happened(a,k) \in \Gamma$;
2. if $obs(l,k) \in \Gamma$ then $l \in \sigma_k$.

If Γ has a model we say that Γ is *consistent* (with respect to D). We say that fluent literal l holds in a model M at time $k \leq n$ (and write $M \models h(l,k)$) if $l \in \sigma_k$. We say $\Gamma \models h(l,k)$ if $h(l,k)$ holds in all models of Γ.

To better understand the terminology let us consider the following example.

Example 0.1 Consider a system S consisting of an analog circuit $\mathcal{AC}$ from figure 1. Switches s_1 and s_2 are mechanical components which cannot become damaged. Relay r is a magnetic coil. If not damaged, it is activated when s_1 is closed causing s_2 to become closed. Undamaged bulb b emits light if s_2 is closed. For simplicity we consider an agent capable of performing one action, $close(s_1)$. The environment can be represented by two damaging exogenous actions: $brks$, which causes b to become faulty, and srg, which damages r and may damage b if b is not protected.

The description, SD, of S consists of statements:

Types
$comp(r). \quad comp(b).$
$switch(s_1). \quad switch(s_2).$

Fluents
$$\begin{array}{lll} f(closed(SW)) & \leftarrow & switch(SW). \\ f(ab(X)) & \leftarrow & comp(X). \\ f(active(re)). & f(prot(b)). & f(on(b)). \end{array}$$

$$AgentActions\ \{\ a\_act(close(s_1)).$$

$$ExogenousActions \left\{ \begin{array}{l} x\_act(brks). \\ x\_act(srg). \end{array} \right.$$

Causal Laws and Executability Conditions describing normal functioning of S are expressed as follows:

$$D_n \left\{ \begin{array}{l} causes(close(s_1), closed(s_1), []). \\ caused(active(r), [closed(s_1), \neg ab(r)]). \\ caused(closed(s_2), [active(r)]). \\ caused(on(b), [closed(s_2), \neg ab(b)]). \\ caused(\neg on(b), [\neg closed(s_2)]). \\ impossible\_if(close(s_1), [closed(s_1)]). \end{array} \right.$$

($causes(A, L, P)$ says that execution of action A in a state satisfying fluent literals from P causes fluent literals L to become true in a resulting state; $caused(L, P)$ means that every state satisfying P must also satisfy L, $impossible\_if(A, P)$ indicates that action A is not executable in states satisfying P.)

The system's malfunctioning information is given by:

$$D_b \left\{ \begin{array}{l} causes(brks, ab(b), []). \\ causes(srg, ab(r), []). \\ causes(srg, ab(b), [\neg prot(b)]). \\ caused(\neg on(b), [ab(b)]). \\ caused(\neg active(r), [ab(r)]). \end{array} \right.$$

Now consider a history, $\Gamma_0(0)$ of S:

$$\Gamma_0(0) \left\{ \begin{array}{l} hpd(close(s_1), 0). \\ obs(\neg closed(s_1), 0). \\ obs(\neg closed(s_2), 0). \\ obs(\neg ab(b), 0). \\ obs(\neg ab(r), 0). \\ obs(prot(b), 0). \end{array} \right.$$

It is easy to see that the path $\sigma_0, close(s_1), \sigma_1$ is the only model, of $\Gamma_0(0)$ w.r.t. SD and that $\Gamma_0(0) \models h(on(b), 1)$

Possible diagnosis

Let $\Gamma(t_c)$ be a history of system S described by SD, $o(t_c + 1)$ be the set of fluent literals observed by the system controlling agent at moment $t_c + 1$, and $O(T) = \{obs(l, T) : l \in o(T)\}$. By a *diagnostic state* of S at moment t_c we mean a pair,

$$DS(t_c) = (\Gamma(t_c), o(t_c + 1)) \quad (1)$$

If S behaves properly then $o(t_c + 1)$ does not contradict $\Gamma(t_c)$, i.e. $\Gamma(t_c + 1) = \Gamma(t_c) \cup O(t_c + 1)$ is consistent with respect to SD. Otherwise we say that $DS(t_c)$ is a *symptom* of the system's malfunctioning and therefore S needs diagnosis.

Our definition of a possible diagnosis is based on the notion of an *explanation* from (Baral & Gelfond 2000). In our terminology, an explanation, E, of a symptom $DS(t_c)$, is a collection of statements

$$E = \{hpd(a_i, t) : t \leq t_c \text{ and } a_i \in A_e\} \quad (2)$$

such that $\Gamma(t_c) \cup O(t_c + 1) \cup E$ is consistent with respect to SD.

Definition A *possible diagnosis* of a symptom $DS(t_c)$ consists of explanation E of DS together with the set Δ of components possibly damaged by actions from E. More precisely, $\Delta = \{c : M \models h(ab(c), t_c)\}$ where M is a model of $\Gamma(t_c) \cup O(t_c + 1) \cup E$ with respect to SD.

To illustrate our definitions let us again consider system S from Example 0.1. According to $\Gamma_0(0)$, at moment 0, s_1 and s_2 are open, all circuit components are ok, s_1 is closed by the agent, b is protected, and b is on at 1. Suppose that instead, the agent observes that at time 1 bulb b is off, i.e. $o(1) = \{\neg on(b)\}$. Intuitively, this is viewed as a symptom, $DS_0(0)$, of the malfunctioning of S. The same conclusion can be obtained formally by checking that $\Gamma_0(0) \cup O(1)$ is inconsistent with respect to SD. It is not difficult to see that there are three possible diagnoses of $DS(0)$:

$D1 = \langle\{hpd(brks, 0)\}, \{b\}\rangle$
$D2 = \langle\{hpd(srg, 0)\}, \{r\}\rangle$
$D3 = \langle\{hpd(brks, 0), hpd(srg, 0)\}, \{b, r\}\rangle$

Computing Diagnoses

In this section we show how the need for diagnosis can be determined and how diagnoses can be found by the techniques of answer set programming. We start by describing an encoding α of causal laws of $\mathcal{AL}_d$ into a program of A-Prolog suitable for execution by SMODELS:

1. $\alpha(causes(a, l_0, [l_1 \dots l_n]))$ is the collection of atoms $d\_law(d)$, $head(d, l_0)$, $action(d, a)$ and $prec(d, l_i)$ for every l_i $(1 \leq i \leq n)$ where $d = d_i(\bar{t})$ with d_i being a new function symbol and $\bar{t}$ being a list of terms occurring in the corresponding causal law.
2. $\alpha(caused(l_0, [l_1 \dots l_n]))$ is the collection of atoms $s\_law(d)$, $head(d, l_0)$, $prec(d, l_i)$ for every l_i $(1 \leq i \leq n)$.
3. $\alpha(impossible\_if(a, [l_1 \dots l_n]))$ is a constraint $\leftarrow h(l_1, T), \dots, h(l_n, T), oc(a, T)$.

Finally, $\alpha(SD) = \{\alpha(ax) : ax \in SD\} \cup \Pi(N)$ where $\Pi(N)$ is the following program:

$$\Pi(N) \left\{ \begin{array}{lll} 1.\ h(L, T') & \leftarrow & d\_law(D), \\ & & head(D, L), \\ & & action(D, A), \\ & & oc(A, T), \\ & & prec\_h(D, T). \\ 2.\ h(L, T) & \leftarrow & s\_law(D), \\ & & head(D, L), \\ & & prec\_h(D, T). \\ 3.\ prec\_f(D, T) & \leftarrow & prec(D, P), \\ & & not\ h(P, T). \\ 4.\ prec\_h(D, T) & \leftarrow & not\ prec\_f(D, T). \\ 5.\ h(L, T') & \leftarrow & f(L), \\ & & h(L, T), \\ & & not\ h(\overline{L}, T'). \\ 6.\ oc(A, T) & \leftarrow & hpd(A, T). \\ 7.\ h(L, T) & \leftarrow & obs(L, T). \end{array} \right.$$

Here D, A, L are variables for the names of laws, actions, and fluent literals respectively, and T, T' denote consecutive time points from some interval $[0, N]$; *oc* which stands for *occurs*, is used to allow separation between actions observed and actions hypothetized. Let

$$\Pi_0(\Gamma(t_c)) = \alpha(SD) \cup \Gamma(t_c) \tag{3}$$

The following terminology will be useful for describing the relationship between answer sets of $\Pi_0(\Gamma(t_c))$ and models of $\Gamma(t_c)$. We say that an answer set $\mathcal{AS}$ of $\Pi_0(\Gamma(t_c))$ *defines* a trajectory $p = \sigma_0, a_0, \sigma_1, \dots, a_{t_c-1}, \sigma_{t_c}$ where, $l \in \sigma_k$ iff $h(l, k) \in \mathcal{AS}$ $(0 \leq k \leq t_c)$ and $a_k = \{a : oc(a, k) \in \mathcal{AS}\}$ $(0 \leq k < t_c)$.

The following important theorem (a variant of the result from (Turner 1997) and the definitions from (McCain & Turner 1995)) establishes the relationship between the theory of actions and logic programming.

Theorem 0.1 *Let $\Gamma(t_c)$ be a history of system S with a complete initial situation. M is a model of $\Gamma(t_c)$ iff M is a trajectory defined by some answer set of $\Pi_0(\Gamma(t_c))$.*

A-Prolog diagnostic algorithms will be based on the corollary of Theorem 0.1. First some notation. Let DS be a diagnostic state of a system S. By $\beta(SD)$ we denote $\alpha(SD)$ combined with the following rules:

$$\begin{array}{lll} h(F, 0) & \leftarrow & not\ h(\neg F, 0). \\ h(\neg F, 0) & \leftarrow & not\ h(F, 0). \\ & \leftarrow & obs(l, t_c + 1), not\ h(l, t_c + 1). \end{array}$$

The first two rules, sometimes called the awareness axioms, say that either F or $\neg F$ should be included in the beliefs of a reasoning agent. If the agent's information about the initial state of the system is complete these axioms can be omitted. The last axiom says that the agent's observations are correct and hence shall coincide with the agent's predictions.

Corollary 0.1 *A diagnostic state $DS(t_c)$ is a symptom of a malfunctioning system S iff a program*

$$TEST(DS) = \beta(SD) \cup \Gamma(t_c) \cup O(t_c + 1) \tag{4}$$

does not have an answer set.

To diagnose a symptom DS we construct a program, DM, which generates possible explanations of DS. Such programs will be termed *diagnostic modules*. Finding diagnoses will be reduced to finding answer sets of the program

$$\mathcal{D}(DS) = TEST(DS) \cup DM \tag{5}$$

The simplest diagnostic module, DM_0, can be defined by the following rules:

$$\begin{array}{lll} 1.\ oc(A, T) & \leftarrow & 0 \leq T \leq t_c, \\ & & x\_act(A), \\ & & not\ \neg oc(A, T). \\ \\ 2.\ \neg oc(A, T) & \leftarrow & 0 \leq T \leq t_c, \\ & & x\_act(A), \\ & & not\ oc(A, T). \end{array}$$

$\mathcal{D}_0 = TEST(DS) \cup DM_0$ finds possible diagnoses of the symptom DS by generating sets of occurrences of exogenous actions and testing if they constitute possible diagnoses. For any positive result, the possible diagnosis can be extracted from the corresponding answer set of $\mathcal{D}_0$. It is not difficult to see that DM_0 generates every possible set of occurrences of exogenous actions and hence, by Theorem 1, $\mathcal{D}_0$ finds all the diagnoses of DS. In our further discussion we will deviate from the strict adherence to the syntax of A-Prolog and use instead a choice operator of SMODELS (Simons 1999). In the new notation, rules (1) and (2) in DM_0 will be replaced by one rule

$$\{oc(A,T) : x\_act(A)\} \quad \leftarrow \quad 0 \le T \le t_c$$

Refining the diagnoses

Normally, we are not interested in all possible diagnoses of a symptom - only in those which satisfy certain criteria. In this section we demonstrate how some of such criteria can be incorporated in the corresponding diagnostic modules. In many cases, for instance, some constant, m, determines the time interval in the past that an agent is willing to consider in it's search for possible explanations. This limiting of the search space can be incorporated in the rules of DM_0 by replacing a statement $0 \le T \le t_c$ in their bodies by $t_c - m \le T \le t_c$. (For simplicity, in the following program examples we let $m = 0$.) Sometimes the search can be limited by restricting attention to explanations consisting of a limited number of actions. This can be easily implemented using the choice rule of SMODELS. For instance, the program $\mathcal{D}_1 = TEST(DS) \cup DM_1$ where DM_1 is

$$1\{oc(A,t_c) : x\_act(A)\}2.$$

will find all diagnoses of a symptom DS which can be explained by at most two occurrences of exogenous actions at moment t_c. For diagnostic symptom DS_0 in the previous section, $\mathcal{D}_1$ will generate diagnoses D1-D3 (with *hpd* replaced by *oc*). It is worth noticing that if we were to expand system description SD from Example 0.1 by adding one more exogenous action, a, completely unrelated to DS_0, $\mathcal{D}_1$ would return additional diagnoses containing a. Diagnoses containing actions irrelevant to the symptom can, in many cases, be eliminated by expanding diagnostic modules with domain independent rules. It would be very natural for instance to write something like:

$$\begin{array}{lcl}
rel(A,L) & \leftarrow & d_law(D), \\
 & & head(D,L), \\
 & & action(D,A), \\
 & & x_act(A). \\
rel(A,L) & \leftarrow & s_law(D), \\
 & & head(D,L), \\
 & & prec(D,P), \\
 & & rel(A,P), \\
 & & x_act(A). \\
1\{oc(A,t_c) : rel(A,L)\}2 & \leftarrow & obs(L,t_c+1).
\end{array}$$

Unfortunately, SMODELS will not allow the use of the last rule since the relation rel is defined recursively. Instead we can replace this rule by

$$\begin{array}{lcl}
\multicolumn{3}{l}{1\{oc(A,t_c) : x_act(A)\}2.} \\
rel(A) & \leftarrow & obs(L,t_c+1), \\
 & & rel(A,L). \\
 & \leftarrow & oc(A,t_c), \\
 & & not\ rel(A).
\end{array}$$

Even if these new rules are incorporated in $\mathcal{D}_1$ it will still return diagnosis D3. It would be very nice to be able to construct a diagnostic module capable of only finding diagnoses minimal with respect to some diagnostic ordering. (A simple set-theoretic ordering would allow us to drop diagnosis D_3 from Example 0.1). Unfortunately, in general, A-Prolog without disjunction is not sufficiently powerful for this purpose. We are however often able to use A-Prolog to express various heuristics which allow us to produce higher quality diagnoses. To illustrate this point let us consider the following example.

Example 0.2 Let us consider system S from Example 0.1 and suppose that in his search for diagnoses the diagnostician of S uses the following information: *Normally, an unexpected absence of light can be explained by a broken bulb*. However, if there was a recent storm in the area the diagnostician uses a different rule which says: *Unexpected absence of light after a storm is normally caused by a power surge*. The diagnostic module modeling our diagnostician consists of the rules:

$$\begin{array}{llcl}
1 & oc(brks,t_c) & \leftarrow & obs(\neg on(b),t_c+1), \\
 & & & not\ \neg oc(brks,t_c), \\
 & & & not\ btr(d1,t_c). \\
2 & f(d1,T) & \leftarrow & obs(\neg on(b),t_c+1), \\
 & & & not\ \neg oc(brks,t_c). \\
 & & & not\ btr(d1,t_c). \\
3 & oc(srg,t_c) & \leftarrow & obs(\neg on(b),t_c), \\
 & & & h(storm,t_c), \\
 & & & not\ \neg oc(srg,t_c), \\
 & & & not\ btr(d2,t_c). \\
4 & f(d2,T) & \leftarrow & obs(\neg on(b),t_c), \\
 & & & h(storm,t_c), \\
 & & & not\ \neg oc(srg,t_c), \\
 & & & not\ btr(d2,t_c). \\
5 & \{oc(A,t_c) : x_act(A)\} & \leftarrow & obs(\neg on(b),t_c+1), \\
 & & & not\ btr(d3,t_c). \\
6 & \neg oc(A,T) & \leftarrow & d_law(D), \\
 & & & comp(C), \\
 & & & head(D,ab(C)), \\
 & & & action(D,A), \\
 & & & obs(\neg ab(C),T). \\
7 & btr(D1,T) & \leftarrow & prefer(D2,D1), \\
 & & & f(D2,T). \\
8 & prefer(d2,d1). & & \\
 & prefer(d1,d3). & & \\
 & prefer(d2,d3). & &
\end{array}$$

Rules (1), (3) of the program represent defaults (d1) and (d2) mentioned above. Default (d3) defined by (5) is a catch-all rule the diagnostician resorts to when the first two defaults are inapplicable. Priorities between defaults are given by the relation $prefer$. Literals of the form $f(i, T)$ indicate that default i was applied in the process of constructing the corresponding answer set. The relation $btr(D, T)$, defined by rule (7), holds when a default preferred to D is fired at moment T. The presence of $not\ \ btr(1, T)$ in the body of rule (1) together with rule (4) guarantees that default $d1$ will not be applied if more specific (and therefore preferred) default $d2$ can be fired instead. (A more natural representation may be obtained by reifying defaults and rules but we hope that the above example is sufficient to illustrate the point). Finally, rule (6) expresses the impossibility of certain actions. It suffices for system descriptions containing only dynamic causal laws for fluents of the form ab(c). (Similar laws can be written for broader classes of domain descriptions).

The corresponding program, $\mathcal{D}_2$ will diagnose the symptom, DS_0, with possible diagnoses, $D1$ and $D2$; $D1$ corresponds to the absence of a storm, while $D2$ reflects the possibility of stormy weather. Now consider a new history, Γ_1, which is obtained from Γ_0 by adding a default:

$$\neg h(storm, 0) \quad \leftarrow \quad not\ h(storm, 0).$$

The new symptom, DS_1, still requires diagnosis. This time $D1$ will (correctly) be the only possible diagnosis found by $\mathcal{D}_2$. (Of course if the diagnostician learns $h(storm, 0)$ he will change his mind and will come up with a new possible diagnosis, $D2$.)

Now let us assume that the diagnostician is capable of testing system components. His possible diagnosis, $D1$, suggests that he should test bulb b. Suppose that the test reveals the bulb is ok. Intuitively, the diagnostician knows that $brks$ did not occur and now must believe that srg occurred. Likewise, if there had been a storm, and testing determined that the relay *re* is ok, the diagnostician now knows that srg did not happen and must that $brks$ occurred. This is exactly the behavior exhibited by $\mathcal{D}_2$. Notice, however, that $\mathcal{D}_2$ does not deal with finding possible diagnoses of observations containing statements of the form $ab(c)$. Again, for many system descriptions, this information can be extracted from the corresponding causal laws. We plan to discuss some possible ways of doing this in the full paper. In our example it will suffice to expand $\mathcal{D}_2$ into $\mathcal{D}_3$ by the rules:

$$\begin{array}{lll}
oc(brks, t_c) & \leftarrow & obs(ab(b), t_c + 1), \\
 & & h(prot(b), t_c). \\
f(d4, t_c) & \leftarrow & obs(ab(b), t_c + 1), \\
 & & h(prot(b), t_c). \\
prefer(d4, di). & (1 \leq i \leq 3) & \\
oc(srg, t_c) & \leftarrow & obs(ab(r), t_c). \\
f(d5, t_c) & \leftarrow & obs(ab(r), t_c). \\
prefer(d5, d1). & (1 \leq i \leq 3) &
\end{array}$$

Testing and repairing components

In this section we discuss a more detailed modeling of a diagnostician capable of testing and repairing simple components. We present an algorithm which will enable a **DPS** to maintain a system.

Definition We say that a *diagnosis* of a symptom $DS(t_c) = (\Gamma(t_c), O(t_c + 1))$ is a possible diagnosis in which all components in Δ are faulty.

A diagnosis of a symptom can be found by a function $FIND\_DIAG$ which is used in the following algorithm, $MAINTAIN$. $MAINTAIN$ monitors a system S's behavior and, if a symptom DS is discovered, repeatedly executes the following three steps until S is no longer malfunctioning or a diagnosis cannot be found:

1. Find a diagnosis
2. Repair faulty components
3. Check the system

A more detailed description is given below:

procedure $MAINTAIN(SD, DS)$;

$\{$ If no diagnosis for symptom DS can be found the procedure exits with failure. Otherwise it restores the correct behavior of the system by fixing the faulty components. The system's state DS is updated to contain a record of the repairing process and a record of observations made at the end of repair. $\}$

```
while NEEDS_DIAG(SD, DS) do
   begin
      if not FIND_DIAG(SD, DS, Δ) then
         EXIT - print("NO DIAGNOSIS") ;
      else
         begin
            REPAIR(Δ);
            O = OS(S) ∪ {obs(¬ab(c), t_c + 1) : c} ∈ Δ}
            Γ := Γ ∪ O;
         end
   end
```

Function $NEEDS\_DIAG$ determines whether or not S is malfunctioning by making a call to SMODELS. $FIND\_DIAG$ also makes a call to SMODELS to get a possible diagnosis containing a set, Δ, of faulty components. It then proceeds to test all components in Δ which have not been tested before. If a suspected faulty component c is tested to be ok, $obs(\neg ab(c), t_c + 1)$ is added to O. $REPAIR$ repairs all faulty components in Δ and OS observes fluent literals of S as $obs(l, t_c + 1)$.

To illustrate algorithm $MAINTAIN$, we again refer to Example 0.2. Let us consider the following scenario: At time 0, b is not protected, there is a storm, all components are ok, both switches are open, and an agent closes s_1. At time 1, our **DPS** observes that b is not on.

¿From function $NEED\_DIAG$, a diagnostic symptom DS is detected. Using $\mathcal{D}_3$, $FIND\_DIAG$ generates possible diagnosis $PD_1 = \langle\{oc(srg, 0)\}, \{r, b\}\rangle$ and determines that PD_1 is not a diagnosis because r is found to be ok. Observation $obs(\neg ab(r), 1)$ is added to O and $\mathcal{D}_3$ generates another possible diagnosis $PD_2 = \langle\{oc(brks, 0)\}, \{b\}\rangle$. Bulb b is then tested by our agent, and is found to be faulty. $FIND\_DIAG$ returns PD_2. Bulb b is repaired and b is observed to be on. S is now functioning correctly.

Finding a diagnosis may not always lead to a correct explanation of a symptom of a malfunctioning system. Let us consider the above scenario again, except that this time there is no storm at time 0. $FIND\_DIAG$ computes PD_2, b is found to be faulty, and $\Delta = \{b\}$ of diagnosis PD_2 is returned. Now suppose b is repaired, but it is observed that b is still not on! Observation $obs(\neg ab(b), 1)$ is added to O and $FIND\_DIAG$ would eventually return Δ of PD_1. In order to always find a correct explanation, it is essential to repair damaged components and observe the behavior of the system.

Conclusion

The paper describes an ongoing work on the development of a diagnostic problem solving agent in A-Prolog. We presented a new definition of a symptom and possible diagnosis based on action description language $\mathcal{AL}_d$ and showed how it can be used to search for possible diagnoses in A-Prolog. Several examples outline the methodology of constructing diagnostic modules containing heuristic information. In particular we showed how such information can be expressed in A-Prolog in the form of prioritized defaults. The last section contains a more general architecture of an agent capable of combining diagnostic computation with testing and repair. In the full paper we plan to give mathematical analysis of correctness of the corresponding algorithms and test them on medium size examples.

References

Baral, C., and Gelfond, M. 1999. Reasoning agents in dynamic domains. In *(to appear) Logic Based Approach to AI* .

Baral, C., and Gelfond, M. 1994. Logic programming and knowledge representation. In *Journal of Logic Programming*, volume 12, 1–80.

Baral, C., Gelfond, M., and Provetti, A. 1994. Reasoning about actions: laws, observations, and hypotheses. In *Journal of Logic Programming*, volume 31, 201–244.

Baral, C., McIlraith, S., and Son, T. 2000. Formulating diagnostic problem solving using an action language with narratives and sensing. In *Proceedings of the 2000 KR Conference*, ???

de Kleer, J., Mackworth, A., and Reiter, R. 1992. Characterizing diagnoses and systems. In *Artificial Intelligence*, volume 56(2-3), 197–222.

Gelfond, M., and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, 1070–1080.

Gelfond, M., and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. In *New Generation Computing*, 365–387.

Gelfond, M., and Lifschitz, V. 1992. Representing actions in extended logic programs. In *Proc. of Joint International Conference and Symposium on Logic Programming*, 559–573.

Gelfond, M., and Lifschitz, V. 1998. Action languages. In *Electronic Transactions on AI*, volume 3(16).

Lifschitz, V. 1999. Action languages, Answer Sets, and Planning. In *The Logic Programming Paradigm: a 25-Year Perspective*.

Marek, W., and Truszczynski, M. 1999. Stable models and an alternative logic paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, 375–398.

McCain, T., and Turner, H. 1995. A causal theory of ramifications and qualifications. In *Artificial Intelligence*, volume 32, 57–95.

Niemela, I. 1996. Logic programs with stable model semantics as a constraint programming paradigm. In *Annals of Mathematics and Artificial Intelligence*.

Niemela, I., and Simons, P. 1996. Efficient implementation of the well-founded and stable model semantics. In *Proc. of Joint Int'l Conf. and Symposium on Logic Programming*, 289–303.

Reiter, R. 1987. A theory of diagnosis from first principles. In *Artificial Intelligence*, volume 32, 57–95.

Simons, P. 1999. Extending the stable model semantics with more expressive rules. In *5th International Conference, LPNMR'99*, 305–316.

Thielscher, M. 1997. A theory of dynamic diagnosis. In *Linkoping Electronic Articles in Computer and Information Science*, volume 2(11). 1

H. Turner. 1997. Representing actions in logic programs and default theories. In *Journal of Logic Programming*, 31(1-3):245–298, May 1997.