From: AAAI Technical Report SS-01-01. Compilation copyright © 2001, AAAI (www.aaai.org). All rights reserved.

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

The main goal of this paper is to continue the investigation of applicability of A-Prolog (a loosely defined collection of logic programming languages under the answer set semantics (Gelfond & Lifschitz 1991)) to knowledge representation and reasoning. In the first part of the paper we address the problem of diagnosing faulty behavior of a dynamic system. We are especially interested in such classical aspects of this problem as modeling of systems and their environments, interleaving of deliberation and action, and prioritizing typical causes of specific faults. We start with suggesting a new version of a definition of diagnosis. The roots of our definition go back to (Reiter 1987) and its recent modifications (Thielscher 1997; Baral, McIlraith & Son 2000) which take into account the dynamics of system's behavior. 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 *domain description* of an action language $\mathcal{AL}$ (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 and allows us to give a definition of diagnosis which, we believe, is substantially simpler than other definitions we were able to find in the literature. 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 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 a set, C, of *components*. We assume that a state of the system can be described by a collection F of *fluents* - relevant properties of these components whose truth values may depend on time, and that the dynamics of the system are controlled by a set $A = A_s \cup A_e$ of (elementary) *actions* capable of changing S's states. The set A_s consists of actions performed by the system (or an agent controlling the system) and a set A_e consists of exogenous actions whose occurrence can cause system components to malfunction. A system S will be modeled by a *system description* SD consisting of rules of A-Prolog defining components of S, its fluent and actions, causal laws determining the effects of these actions, and the actions' executability conditions. A state of S will be identified with a complete and consistent set of fluent literals which are true in this state. (By fluent literals we mean fluents and their negations.) We assume that SD has a unique answer set which defines an action description of $\mathcal{AL}$. (In our further discussion we will often abuse the notation and identify this action description with SD.) Semantics of $\mathcal{AL}$ associates SD 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. Causal laws of SD can be divided into two parts. The first part, SD_n, contains laws describing normal behavior of the system. Their bodies usually contain special fluent lit-

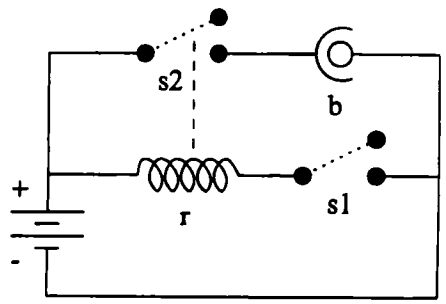

Figure 1: $\mathcal{AC}$

erals 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, SD_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 up to a moment n is specified by a collection Γ_n of statements in the 'history description' part of $\mathcal{AL}$. We assume that the system's time is discrete, t_i and t_{i+1} stand for two consecutive moments of time in the interval $0..n$. Statements of Γ have the form:

1. $obs(l,t)$ (fluent literal l was observed to be true at moment t), and
2. $hpd(a,t)$ (elementary action $a \in A$ was observed to happen at moment t) where $\leq t < n$.

To simplify the presentation we *only consider histories with observations closed under the static causal rules of $\mathcal{AL}$*, (i.e. if every state of S must satisfy a constraint 'fluent literal l_0 is true if fluent literals from P are true' and literals from P are observed in Γ the so must be l_0).

The following definitions are from (Baral & Gelfond 2000). Let SD 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. SD iff

1. $a_k = \{a : hpd(a,k) \in \Gamma$;
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 SD). 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. Notice that, in contrast to $\mathcal{L}$, a domain description of $\mathcal{AL}$ is consistent only if changes in the observations of system's states can be explained without assuming occurrences of any action not recorded in 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 b if b is not protected.

The description, SD, of S consists of statements:

Components

$comp(r)$. $comp(b)$.
$switch(s_1)$. $switch(s_2)$.

Fluents

$f(closed(SW)) \leftarrow switch(SW)$.
$f(ab(X)) \leftarrow comp(X)$.
$f(active(re))$. $f(prot(b))$. $f(on(b))$.

Notice that this description implies that, according to our model, switches never malfunction.

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

$ExogenousActions$ { $x\_act(brks)$. $x\_act(srg)$.

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

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

($causes(A, L, P)$ 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:

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

Now consider a history, Γ_0 of S:

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

It is easy to see that the path $\sigma_0, close(s_1), \sigma_1$ is the only model, of Γ_0 w.r.t. SD and that $\Gamma_0 \models h(on(b), 1)$

Possible diagnosis

Let Γ_{n-1} be a history of system S described by SD, o_n be the set of fluent literals observed by the system controlling agent at moment n, and $O_T = \{obs(l,T) : l \in o_T\}$. By a *diagnostic state* of S at moment n we mean a pair,

$$DS_n = (\Gamma_{n-1}, o_n) \quad (1)$$

If S behaves properly then o_n does not contradict Γ_{n-1}, i.e. $\Gamma_n = \Gamma_{n-1} \cup O_n$ is consistent with respect to SD. Otherwise we say that DS_n is a *symptom* of the system's malfunctioning at moment n 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_n, is a collection of statements

$$E = \{hpd(a_i,t) : 0 \leq t < n \text{ and } a_i \in A_e\} \quad (2)$$

such that $\Gamma_{n-1} \cup O_n \cup E$ is consistent with respect to SD.

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

Example 0.2 To illustrate our definitions let us again consider system S from Example 0.1. According to Γ_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_1, of the malfunctioning of S. The same conclusion can be obtained formally by checking that $\Gamma_0 \cup O_1$ is inconsistent with respect to SD. It is not difficult to see that there are three possible diagnoses of DS_1:

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

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 domain descriptions of $\mathcal{AL}$ into programs of A-Prolog suitable for execution by SMODELS. Since SMODELS takes as an input programs with finite Herbrand base we first need to eliminate the references to lists in causal laws of $\mathcal{A}$. This can be done as follows:

1. $\alpha(causes(a, l_0, [l_1 \ldots l_n]))$ 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. $\alpha(caused(l_0, [l_1 \ldots l_n]))$ 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. $\alpha(impossible\_if(a, [l_1 \ldots l_n]))$ is a constraint
   $\leftarrow$ $h(l_1, T), \ldots, h(l_n, T)$, $oc(a, T)$.

Now consider domain description of $\mathcal{AL}$ with action description SD, history Γ_n, and let

$$\alpha(SD, \Gamma) = \{\alpha(ax) : ax \in SD\} \cup \Pi(n)$$

where $\Pi(n)$ is defined as follows:

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

Here D, A, L are variables for the names of laws, actions, and fluent literals respectively, and T, T' denote consecutive time points from interval $[0, n]$; oc, which stands for *occurs*, is used to allow separation between actions observed and actions hypothesized. The following terminology will be useful for describing the relationship between answer sets of $\alpha(SD, \Gamma)$ and models of Γ with respect to SD:

We say that an answer set $\mathcal{AS}$ of $\alpha(SD, \Gamma_n)$ *defines* a trajectory $p = \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n$ where, $l \in \sigma_k$ iff $h(l,k) \in \mathcal{AS}$ $(0 \leq k \leq n)$ and $a_k = \{a : oc(a,k) \in \mathcal{AS}\}$ $(0 \leq k < n)$.

The following theorem establishes the relationship between the theory of actions in $\mathcal{AL}$ and logic programming.

Theorem 0.1 *Let SD be a description of system S and Γ be its history up to the moment n. If the initial situation of Γ is complete (i.e. for any fluent f of SD, Γ contains $obs(f,0)$ or $obs(\neg f,0)$) then M is a model of Γ iff M is a trajectory defined by some answer set of $\alpha(SD,\Gamma)$.*

(The theorem is similar to the result from (Turner 1997) which deals with a different language and uses the definitions from (McCain & Turner 1995)). The following corollary will be useful for some diagnostic algorithms discussed in this paper. First some notation. Let SD

be a description of system S, $DS = (\Gamma_n, o_{n+1})$ be a diagnostic state of S,

$$TEST(DS) = \alpha(SD, \Gamma) \cup O_{n+1} \cup R_1 \cup R_2 \quad (3)$$

where

$$R_1 \begin{cases} h(F,0) & \leftarrow \ not\ h(\neg F, 0). \\ h(\neg F,0) & \leftarrow \ not\ h(F,0). \end{cases}$$

and

$$R_2 \ \{ \quad \leftarrow \ obs(l, n+1), not\ h(l, n+1).$$

The rules of R_1 are sometimes called the *awareness axioms*. They guarantee that initially the agent considers all possible values of the domain fluents. If the agent's information about the initial state of the system is complete these axioms can be omitted. The R_2 says that the agent's observations are correct and hence shall coincide with the agent's predictions.

Corollary 0.1 *A diagnostic state DS is a symptom of a malfunctioning system S iff the program $TEST(DS)$ has no 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

$$\mathcal{D}(DS) = TEST(DS) \cup DM \quad (4)$$

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

$$1.\ oc(A,T) \leftarrow 0 \le T \le n, \ x\_act(A), \ not\ \neg oc(A,T).$$

$$2.\ \neg oc(A,T) \leftarrow 0 \le T \le n, \ x\_act(A), \ not\ oc(A,T).$$

Answer sets of a program

$$\mathcal{D}_0 = TEST(DS) \cup DM_0$$

correspond to possible diagnoses of the symptom DS. SMODELS will find these answer sets if they exist or report failure otherwise. The possible diagnosis can be easily extracted from the corresponding answer set. 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

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

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 $0 \le T \le n$ in their bodies by $n - m \le T \le n$. (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

$$DM_1 = 1\{oc(A,n) : x\_act(A)\}2.$$

will find all diagnoses of a symptom DS which can be explained by at most two occurrences of exogenous actions at moment n. 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:

$$1 \quad rel(A,L) \leftarrow d\_law(D), \ head(D,L), \ action(D,A), \ x\_act(A).$$

$$2 \quad rel(A,L) \leftarrow s\_law(D), \ head(D,L), \ prec(D,P), \ rel(A,P), \ x\_act(A).$$

and

$$3a \quad 1\{oc(A, n-1) : rel(A,L)\}2 \quad \leftarrow \quad obs(L,n).$$

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

$$3b \quad 1\{oc(A, n-1) : x\_act(A)\}2.$$

$$4 \quad rel(A) \leftarrow obs(L,n), \ rel(A,L).$$

$$5 \quad \leftarrow oc(A, n-1), \ not\ rel(A).$$

The new diagnostic module,

$$\mathcal{D}_2 = TEST(DS) \cup DM_2$$

where DM_2 consists of rules (1) – (5) above will eliminate diagnoses containing irrelevant action a. Notice, that the ability of A-Prolog to represent recursive rules together with a one-directional nature of causal laws are responsible for the relative ease of representation.

Unfortunately, the new diagnostic module, 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 0.3 Let us consider a system description SD 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.* These are typical prioritized default rules. Since the second rule is more specific if should be preferred to the first one whenever possible. The diagnostic module DM_3 modeling our diagnostician is built according to a general methodology of representing prioritized defaults in A-Prolog. it consists of the rules:

1 $oc(brks, n-1) \leftarrow obs(\neg on(b), n),$ $not\ \neg oc(brks, n-1),$ $not\ better(d1, n-1).$

2 $f(d1, T) \leftarrow obs(\neg on(b), n),$ $not\ \neg oc(brks, n-1).$ $not\ better(d1, n-1).$

3 $oc(srg, n-1) \leftarrow obs(\neg on(b), n),$ $h(storm, n-1),$ $not\ \neg oc(srg, n-1),$ $not\ better(d2, n-1).$

4 $f(d2, T) \leftarrow obs(\neg on(b), n),$ $h(storm, n-1),$ $not\ \neg oc(srg, n-1),$ $not\ better(d2, n-1).$

5 $\{oc(A, n-1) : x\_act(A)\} \leftarrow obs(\neg on(b), n),$ $not\ better(d3, n-1).$

6 $better(D1, T) \leftarrow prefer(D2, D1),$ $f(D2, T).$

7 $prefer(d2, d1).$ $prefer(d1, d3).$ $prefer(d2, d3).$

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 $f(i, T)$ (where f stands for $fired$) indicate that default i was applied in the process of constructing the corresponding answer set. The relation $better(D, T)$, defined by rule (7), holds when a default preferred to D is fired at moment T. The presence of $not\ better(1, T)$ 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. (Arguably, 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).

Notice, that the initial situation in Example 0.1 is incomplete and hence the diagnostic module

$$\mathcal{D}_3 = TEST(DS) \cup DM_3$$

will diagnose the symptom, DS_0, with two possible diagnoses, $D1$ and $D2$; $D1$ corresponds to the absence of a storm, while $D2$ reflects the possibility of stormy weather. If we consider a different history, Γ_0', obtained from Γ_0 by adding

$$\neg h(storm, 0)$$

the new symptom, DS' will still requires diagnosis. This time $D1$ will (correctly) be the only possible diagnosis found by $\mathcal{D}_3$. (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 believe that $brks$ occurred. This is exactly the behavior exhibited by $\mathcal{D}_3$. Notice, however, that by construction, $\mathcal{D}_3$ only finds possible diagnosis of an unexpected absence of light. It does not deal with observations containing statements of the form $ab(c)$. For our example the corresponding diagnostic module can be obtained by expanding $\mathcal{D}_3$ into $\mathcal{D}_4$ by the rules:

$oc(brks, n-1) \leftarrow obs(ab(b), n),$ $h(prot(b), n-1).$

$f(d4, n-1) \leftarrow obs(ab(b), n),$ $h(prot(b), n-1).$

$prefer(d4, di).$ $(1 \leq i \leq 3)$

$oc(srg, n-1) \leftarrow obs(ab(r), n).$

$f(d5, n-1) \leftarrow obs(ab(r), n).$

$prefer(d5, di).$ $(1 \leq i \leq 3)$

For many system descriptions, these types of rules can be extracted automatically from the corresponding

causal laws. We plan to discuss some possible ways of doing this in the full paper.

Testing and repairing components

Let us now assume that a diagnostician is also capable of testing if a given component is faulty and that the tests are performed instantaneously. The last condition is equivalent to the assumption that no actions, exogenous or otherwise, which can change the state of the system are performed during the process of testing. First we need the following definition:

Definition We say that a *diagnosis* of a symptom $DS_n = (\Gamma_{n-1}, o_n)$ is a possible diagnosis in which all components in Δ are faulty.

Let us assume that we have a function $POSSIBLE\_DIAG(SD, DS)$ which returns a possible diagnosis (E, Δ) of a symptom DS of SD. ($\Delta = \emptyset$ indicates that no such diagnosis can be found). The function, $FIND\_DIAG$ which takes a system description SD and a symptom DS_n as parameters and returns a diagnosis $D = (E, \Delta)$ of DS_n can be defined as follows:

REPEAT
  $(E, \Delta) := POSSIBLE\_DIAG(SD, DS)$
  REPEAT
    select $c \in \Delta$;
    if c is faulty
    $\Gamma_{n-1} := \Gamma_{n-1} \cup obs(ab(c, n-1))$;
    else $\Gamma_{n-1} := \Gamma_{n-1} \cup \neg obs(ab(c, n-1))$;
  UNTIL c is not faulty or
    all components are faulty;
UNTIL every element of Δ is faulty;
RETURN (E, Δ).

Notice that $\Delta = \emptyset$ indicates that no diagnosis is found - the diagnostician failed. To illustrate the algorithm, consider

Example 0.4 Consider SD from Example 0.3 and the following initial situation: 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 diagnostician observes that the bulb b is not on. He calls a function $NEED\_DIAG(SD, DS_1)$ which searches for an answer set of $TEST(DS)$. There are no such sets and hence a diagnostic symptom DS is detected. Now the agent calls a function $POSSIBLE\_DIAG$. Let us assume that it is based on diagnostic module $\mathcal{D}_4$. The function returns possible diagnosis

$$PD_1 = \langle \{oc(srg, 0)\}, \{r, b\} \rangle$$

The agent selects a component r from Δ and determines that PD_1 is not a diagnosis because r is found to be ok. Observation $obs(\neg ab(r), 0)$ is added to Γ_0 and $POSSIBLE\_DIAG$ is called again with new DS_1 as a parameter. It generates another possible diagnosis

$$PD_2 = \langle \{oc(brks, 0)\}, \{b\} \rangle$$

Bulb b is then tested by the agent, and is found to be faulty. Observation $obs(ab(b), 0)$ is added to Γ_0 and $FIND\_DIAG$ returns PD_2.

The next example shows that finding a diagnosis may not always lead to a correct explanation of a symptom of a malfunctioning system.

Example 0.5 Let us consider a scenario from Example 0.4 again, except that this time there is no storm at time 0. $FIND\_DIAG$ computes PD_2, b is found to be faulty, $obs(ab(b), 0)$ is added to the history, and diagnosis PD_2 is returned. Now suppose b is repaired, but it is observed that b is still not on! Since we assume that no exogenous actions occur during testing this can mean only that $brks$ is a wrong explanation - we really had a power surge. *In order to always find a correct explanation, it is essential to repair damaged components and observe the behavior of the system after repair.*

This can be done by introducing a special function, $repair(C)$ for every component c of SD. The effect of this action is defined by the following causal law:

$$causes(repair(C), \neg ab(C), [])$$

To make this work we need to modify some of our definitions. Instead of observations o_n in the definition of a diagnostic situation DS_n we must allow observations at any time $T \geq n$. The corresponding history should be allowed to contain the repair actions at these times. Finally, the diagnostic modules must be modified to deal with more complex observations. We will illustrate such a modification by showing how to change rule 1 of DM_3. The new rule will look as follows:

$$\begin{aligned} 1 \quad oc(brks, n-1) \leftarrow\ & time(T), \\ & T \geq n, \\ & obs(\neg on(b), T), \\ & not\ \neg oc(brks, n-1), \\ & not\ better(d1, n-1). \end{aligned}$$

Other modifications are equally natural.

The resulting diagnostic modulus can be used in algorithm $MAINTAIN$ which monitors a system 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
    (E, Δ) = FIND_DIAG(SD, DS);
    if Δ) = ∅ then
      EXIT - print("NO DIAGNOSIS") ;
    else
      begin
        REPAIR(Δ);
        Expand DS by the
        record of repair process;
      end
  end
```

This algorithm will allow us to correctly formalize reasoning from the second scenario of Example 0.5. After the corresponding repair information is recorded in DS the system will find a correct diagnosis PD_1.

Conclusion

The paper describes an ongoing work on the development of a diagnostic problem solving agent in A-Prolog. The work is not intended to be practical - we are still looking for good modeling techniques with clear and provenly correct algorithms. The following can be of interest to people who share these interests:

We presented a new definition of a symptom and possible diagnosis based on action description language $\mathcal{AL}$ 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. 2000. Reasoning agents in dynamic domains. In Minker, J,. ed., *Logic-Based Artificial Intelligence*, Kluwer Academic Publishers, 257–279,

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*, 311-322

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*. 357–373, Springer Verlag, 1999

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

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*, 1999

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.