Diagnosing physical systems in A-Prolog

Michael Gelfond, Marcello Balduccini[1], and Joel Galloway[2]

[1] Department of Computer Science, Texas Tech University
Lubbock, TX 79409, USA
{mgelfond,balduccini}@cs.ttu.edu
WWW home page: http://www.cs.ttu.edu/~mgelfond
[2] Dupont Pharmaceuticals, San Diego, CA, USA
Joel.R.Galloway@dupontpharma.com

Abstract. In this paper we suggest an architecture for a software agent which operates a physical device and is capable of making observations and of testing and repairing the device components. We present novel definitions of the notions of symptom, candidate diagnosis, and diagnosis which are based on the theory of action language $\mathcal{AL}$. The new definitions allow one to give a simple account of the agent's behavior in which many of the agent's tasks are reduced to computing stable models of logic programs.

1 Introduction

In this paper we continue the investigation of applicability of A-Prolog (a loosely defined collection of logic programming languages under the answer set semantics [6]) to knowledge representation and reasoning. The focus is on the development of an architecture for a software agent acting in a changing environment. We assume that the agent and the environment (sometimes referred to as a dynamic system) satisfies the following simplifying conditions.

1. The agent's environment is viewed as a transition diagram whose states are sets of fluents (relevant properties of the domain whose truth values may depend on time) and whose arcs are labeled by actions.
2. The agent is capable of making correct observations, performing actions, and remembering the domain history.

These assumptions hold in many realistic domains and are suitable for a broad class of applications. In many domains, however, the effects of actions and the truth values of observations can only be known with a substantial degree of uncertainty which cannot be ignored in the modeling process. It remains to be seen if some of our methods can be made to work in such situations. The above assumptions determine the structure of the agent's knowledge base. It consists of three parts. The *first part*, called an *action* (or *system*) *description*, specifies the transition diagram representing possible trajectories of the system. It contains descriptions of domain's actions and fluents, together with the definition of possible successor states to which the system can move after an action a is

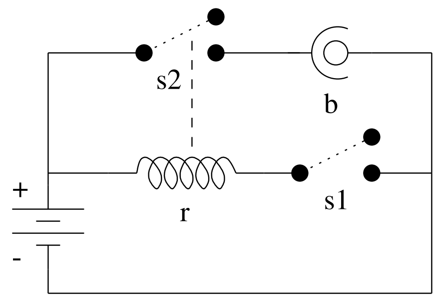

Fig. 1. $\mathcal{AC}$

executed in a state σ. The *second part* of the agent's knowledge, called *history description*, contains observations made by the agent together with a record of its own actions. It defines a collection of paths in the diagram which can be interpreted as the system's possible pasts. If the agent's knowledge is complete (i.e., it has complete information about the initial state and the occurrences of actions) and the system's actions are deterministic then there is only one such path. The *third part* of agent's knowledge base contains a collection of the agent's goals. All this knowledge is used and updated by the agent who repeatedly executes the following steps:

1. observe the world and interpret the observations;
2. select a goal;
3. plan;
4. execute part of the plan.

In this paper we concentrate on agents operating physical devices and capable of testing and repairing the device components. We are especially interested in the first step of the loop, i.e. in agent's interpretations of discrepancies between agent's predictions and the system's actual behavior. The following example will be used throughout the paper:

Example 1. Consider a system S consisting of an analog circuit $\mathcal{AC}$ from figure 1. We assume that 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 close. Undamaged bulb b emits light if s_2 is closed. For simplicity we consider an agent capable of performing only one action, $close(s_1)$. The environment can be represented by two damaging exogenous actions: brk, which causes b to become faulty, and srg, which damages r and also b assuming that b is not protected. Suppose that the agent operating this device is given a goal of lighting the bulb. He realizes that this can be achieved by closing the first switch, performs the operation, and discovers that the bulb is not lit. The goal of the paper is to specify the agent's behavior after this discovery.

We start with presenting our definitions of the notions of symptom, candidate diagnosis, and diagnosis which are based on the theory of action language $\mathcal{AL}$

[1]. These definitions are used to give a simple account of the agent's behavior in which many of the agent's tasks are reduced to computing stable models of logic programs.

Background

By a physical system S we mean a triple $\langle C, F, A\rangle$ of finite sets. Elements of C are called *components* of S. Elements of F are referred to as *fluents*. By *fluent literals* we mean fluents and their negations (denoted by $\neg f$). The set A of *elementary actions* is partitioned into two disjoint sets, A_s and A_e; A_s consists of actions performed by an agent and A_e consists of exogenous actions whose occurrence can cause system components to malfunction.

A system S will be associated with the transition diagram $T(S)$ (or simply T). States of T are labeled by complete and consistent sets of fluent literals corresponding to possible physical states of S. The arcs are labeled by subsets of A called *compound actions*. Execution of a compound action $\{a_1, \ldots, a_k\}$ corresponds to the simultaneous execution of its components. Paths of T correspond to possible behaviors (or trajectories) of S. To reason about S we need to have a concise and convenient way to define its transition diagram. This will be done by a system description $SD(S)$ (or simply 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*. We assume that SD has a unique answer set which defines an action description of $\mathcal{AL}$. (In our further discussion we will identify this action description with SD.) 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 literals of the form $\neg ab(c)$. As usual $ab(c)$ is read as "component c of S is abnormal". Its use in diagnosis goes back to [15]. 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.

In addition to describing all possible trajectories of S, we need to describe the history of S up to a current moment n. This is done by a collection Γ_n of statements in the 'history description' part of $\mathcal{AL}$. We assume that the system's time is discrete and t_i and t_{i+1} stand for two consecutive moments of time in the interval $0 \ldots n$. Statements of Γ_n have the form:

1. $obs(l, t)$ - 'fluent literal l was observed to be true at moment t';
2. $hpd(a, t)$ - elementary action $a \in A$ was observed to happen at moment t

where $0 \leq t < n$. For simplicity 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 then so must be l_0). Let S be a system with the transition diagram T and let Γ_n be a history of S up to moment n. A path $\sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n$ in T is a *model* of Γ_n iff

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

Γ_n is *consistent* (with respect to T) if it has a model. A fluent literal l holds in a model M at time $k \leq n$ ($M \models h(l, k)$) if $l \in \sigma_k$. Finally, $\Gamma_n \models h(l, k)$ if, for every model M of Γ_n, $M \models h(l, k)$. Notice that, in contrast to action description language $\mathcal{L}$ from [2], [3] 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_n.

The following is a description, SD, of system S from Example 1:

Fluents:

$$
\begin{array}{llll}
comp(r). & comp(b). & switch(s_1). & switch(s_2) \\
f(active(r)). & f(on(b)). & f(prot(b)). & \\
f(closed(SW)) \leftarrow & & switch(SW). & \\
f(ab(X)) & \leftarrow & comp(X). &
\end{array}
$$

Agent Actions:	Exogenous Actions
$a\_act(close(s_1))$.	$x\_act(brk)$.
	$x\_act(srg)$.

Causal Laws and Executability Conditions describing normal functioning of S:

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

($causes(A, L, P)$ says that execution of action A in a state satisfying fluent literals from P causes fluent literal 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:

$$
SD_b \begin{cases}
causes(brk, ab(b), []). \\
causes(srg, ab(r), []). \\
causes(srg, ab(b), [\neg prot(b)]). \\
caused(\neg on(b), [ab(b)]). \\
caused(\neg active(r), [ab(r)]).
\end{cases}
$$

Now consider a history, Γ_0 of S:

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

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

2 Basic Definitions

Let S be a system with the transition diagram T, n be a moment of time, O_n be a collection of observations made by the agent starting at n, and Γ_{n-1} be the previous history of S. We say that a pair

$$\mathcal{S} = \langle \Gamma_{n-1}, O_n \rangle \tag{1}$$

is a *symptom* of the system's malfunctioning if Γ_{n-1} is consistent (w.r.t. T) and $\Gamma_{n-1} \cup O_n$ is not. Our definition of a candidate diagnosis of symptom (1) is based on the notion of *explanation* from [1]. In our terminology, an explanation, E, of symptom (1) is a collection of statements

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

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

Definition 1. A *candidate diagnosis* D of symptom (1) consists of an explanation $E(D)$ of (1) together with the set $\Delta(D)$ of components of S which could possibly be damaged by actions from $E(D)$. More precisely, $\Delta(D) = \{c : M \models h(ab(c), n-1)\}$ for some model M of $\Gamma_{n-1} \cup O_n \cup E(D)$.

Definition 2. We say that a *diagnosis* of a symptom $\mathcal{S} = (\Gamma_{n-1}, O_n)$ is a candidate diagnosis in which all components in Δ are faulty.

3 Computing candidate diagnoses

In this section we show how the need for diagnosis can be determined and candidate diagnoses found by the techniques of answer set programming [10].

Consider a system description SD of S whose behavior up to the moment $n-1$ from some interval $[0, N]$ is described by history Γ_{n-1}. (We assume that N is sufficiently large for our application.) We start by describing an encoding of SD into programs of A-Prolog suitable for execution by SMODELS [14]. Since SMODELS takes as an input programs with finite Herbrand bases, references to lists should be eliminated from SD. To do that we expand the signature of SD by new terms - names of the corresponding causal laws - and consider a mapping α defined as follows:

1. $\alpha(causes(a, l_0, [l_1 \ldots l_m]))$ is the collection of atoms $d\_law(d)$, $head(d, l_0)$, $action(d, a)$, $prec(d, i, l_i)$ for $1 \leq i \leq m$, and $prec(d, m+1, nil)$ (Here and below d will refer to the name of the corresponding law).
2. $\alpha(caused(l_0, [l_1 \ldots l_m]))$ is the collection of atoms $s\_law(d)$, $head(d, l_0)$, $prec(d, i, l_i)$ for $1 \leq i \leq m$, and $prec(d, m+1, nil)$.

3. $\alpha(impossible\_if(a, [l_1 \dots l_m]))$ is a constraint

$$\begin{array}{l} \leftarrow h(l_1,T),\dots,h(l_n,T), \\ \quad o(a,T). \end{array}$$

where $o(a,t)$ stands for *action a occurred at time t.*

By $\alpha(SD)$ we denote the result of applying α to the laws of SD. Finally, for any history, Γ, of S

$$\alpha(SD, \Gamma) = \Pi \cup \alpha(SD) \cup \Gamma$$

where Π is defined as follows:

$$\Pi \begin{cases} 1.\ h(L,T') & \leftarrow d\_law(D), \\ & \quad head(D,L), \\ & \quad action(D,A), \\ & \quad o(A,T), \\ & \quad prec\_h(D,T). \\ 2.\ h(L,T) & \leftarrow s\_law(D), \\ & \quad head(D,L), \\ & \quad prec\_h(D,T). \\ 3.\ all\_h(D,N,T) & \leftarrow prec(D,N,nil). \\ 4.\ all\_h(D,N,T) & \leftarrow prec(D,N,P), \\ & \quad h(P,T), \\ & \quad all\_h(D,N',T). \\ 5.\ prec\_h(D,T) & \leftarrow all\_h(D,1,T). \\ 6.\ h(L,T') & \leftarrow h(L,T), \\ & \quad not\ h(\overline{L},T'). \\ 7.\ o(A,T) & \leftarrow hpd(A,T). \\ 8.\ h(L,0) & \leftarrow obs(L,0). \\ 9. & \leftarrow obs(L,T), \\ & \quad not\ h(L,T). \end{cases}$$

Here D, A, L are variables for the names of laws, actions, and fluent literals respectively, T, T' denote consecutive time points from the interval $[0, N]$, and N, N' are variables for consecutive integers. (The corresponding typing predicates in the bodies of some rules of Π are omitted to save space; o is used instead of hpd to distinguish between actions observed and actions hypothesized). The following terminology will be useful for describing the relationship between answer sets of $\alpha(SD, \Gamma_{n-1})$ and models of Γ_{n-1}.

We say that an answer set $\mathcal{AS}$ of $\alpha(SD, \Gamma_{n-1})$ *defines* the trajectory $p = \sigma_0, a_0, \sigma_1, \dots, a_{n-2}, \sigma_{n-1}$ where $\sigma_k = \{l : h(l,k) \in \mathcal{AS}\}$ and $a_k = \{a : o(a,k) \in \mathcal{AS}\}$.

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

Theorem 1. *If the initial situation of Γ_{n-1} is* complete*, i.e. for any fluent f of SD, Γ_{n-1} contains $obs(f,0)$ or $obs(\neg f,0)$ then M is a model of Γ_{n-1} iff M is a trajectory defined by some answer set of $\alpha(SD, \Gamma_{n-1})$.*

(The theorem is similar to the result from [18] which deals with a different language and uses the definitions from [11]).

Now let $\mathcal{S}$ be a symptom of the form (1), and let

$$TEST(\mathcal{S}) = \alpha(SD, \Gamma_{n-1}) \cup O_n \cup R \tag{3}$$

where

$$R \begin{cases} obs(f,0) & \leftarrow not\ obs(\neg f, 0). \\ obs(\neg f, 0) & \leftarrow not\ obs(f, 0). \end{cases}$$

for any fluent $f \in F$. The rules of R 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 following corollary forms the basis for our diagnostic algorithms.

Corollary 1. *Let $\mathcal{S} = \langle \Gamma_{n-1}, O_n \rangle$ where Γ_{n-1} is consistent. Then $\mathcal{S}$ is a symptom of system's malfunctioning iff the program $TEST(\mathcal{S})$ has no answer set.*

To diagnose the system, S, we construct a program, DM, defining an *explanation space* of our diagnostic agent - a collection of sequences of exogenous events which could happen (unobserved) in the system's past and serve as possible explanations of unexpected observations. We call such programs *diagnostic modules* for S. The simplest diagnostic module, DM_0, is defined by rules:

$$DM_0 \begin{cases} o(A,T) & \leftarrow 0 \leq T < n,\ x\_act(A), \\ & \qquad not\ \neg o(A,T). \\ & \\ \neg o(A,T) & \leftarrow 0 \leq T < n,\ x\_act(A), \\ & \qquad not\ o(A,T). \end{cases}$$

or, in the more compact, *choice rule*, notation of SMODELS ([16])

$$\{o(A,T) : x\_act(A)\} \leftarrow 0 \leq T < n.$$

(Recall that a choice rule has the form

$$m\{p(\overline{X})\ :\ q(\overline{X})\}n \leftarrow body$$

and says that, if the body is satisfied by an answer set AS of a program then AS must contain between m and n atoms of the form $p(\bar{t})$ such that $q(\bar{t}) \in AS$.)

Finding candidate diagnoses of symptom (1) can be reduced to finding answer sets of a *diagnostic program*

$$\mathcal{D}_0(\mathcal{S}) = TEST(\mathcal{S}) \cup DM_0 \tag{4}$$

It is not difficult to see that DM_0 generates every possible sequence of the past occurrences of exogenous actions and hence, by Theorem 1, $\mathcal{D}_0(\mathcal{S})$ finds all the candidate diagnoses of $\mathcal{S}$.

Example 2. Let us again consider system S from Example 1. According to Γ_0 initially the switches s_1 and s_2 are open, all circuit components are ok, s_1 is closed by the agent, and b is protected. It is predicted that b will be *on* at 1. Suppose that, instead, the agent observes that at time 1 bulb b is *off*, i.e. $O_1 = \{obs(\neg on(b), 1)\}$. Intuitively, this is viewed as a symptom $\mathcal{S}_0 = \langle \Gamma_0, O_1 \rangle$ of malfunctioning of S. By running SMODELS on $TEST(\mathcal{S}_0)$ we discover that this program has no answer sets and therefore, by corollary 1, $\mathcal{S}_0$ is indeed a symptom. Diagnoses of $\mathcal{S}_0$ can be found by running SMODELS on $\mathcal{D}_0(\mathcal{S}_0)$ and extracting the necessary information from the computed answer sets. It is easy to check that, as expected, there are three candidate diagnoses:

$D_1 = \langle \{o(brk, 0)\}, \{b\} \rangle$
$D_2 = \langle \{o(srg, 0)\}, \{r\} \rangle$
$D_3 = \langle \{o(brk, 0), o(srg, 0)\}, \{b, r\} \rangle$

which corresponds to our intuition. Theorem 1 guarantees correctness of this computation.

The basic diagnostic module $\mathcal{D}_0$ can be modified in many different ways. For instance, a simple modification, $\mathcal{D}_1(\mathcal{S})$ which eliminates some candidate diagnoses containing actions unrelated to the corresponding symptom can be constructed as follows: Let

$$DM_1 = DM_0 \cup REL$$

where

$$REL \left\{ \begin{array}{lll} 1. & rel(A, L) & \leftarrow d\_law(D), \\ & & \quad head(D, L), \\ & & \quad action(D, A), \\ & & \quad x\_act(A). \\ 2. & rel(A, L) & \leftarrow s\_law(D), \\ & & \quad head(D, L), \\ & & \quad prec(D, P), \\ & & \quad rel(A, P), \\ & & \quad x\_act(A). \\ 3. & rel(A) & \leftarrow obs(L, T), \\ & & \quad T \geq n, \\ & & \quad rel(A, L). \\ 4. & & \leftarrow T < n, \\ & & \quad o(A, T), \\ & & \quad not\ hpd(A, T), \\ & & \quad not\ rel(A). \end{array} \right.$$

and let

$$\mathcal{D}_1(\mathcal{S}) = TEST(\mathcal{S}) \cup DM_1$$

It is easy to see that this modification is *safe*, i.e. $\mathcal{D}_1$ will not miss any useful predictions about the malfunctioning components.[1]

[1] In the full paper we will make this and other similar statements mathematically precise.

Example 3. Let us expand the system S from Example 1 by a new component, c, unrelated to the circuit, and an exogenous action a which damages this component. It is easy to see that diagnosis $\mathcal{S}_0$ from Example 1 will still be a symptom of malfunctioning of a new system, S_a, and that the basic diagnostic module applied to S_a will return diagnoses $D_1 - D_3$ from Example 2 together with new diagnoses containing a and $ab(c)$, e.g.

$$D_4 = \langle \{o(brks, 0), o(a, 0)\}, \{b, c\} \rangle$$

Diagnostic module $\mathcal{D}_1$ will ignore actions unrelated to $\mathcal{S}$ and return only $D_1 - D_3$.

It may be worth noticing that the distinction between hpd and o allows actions unrelated to observations at n to actually happen at moment $n-1$. Constraint (4) of REL only prohibits generating such actions in our search for diagnosis. Even more unrelated actions can be eliminated from the search space of our diagnostic modules by considering relevance relation rel depending on time. The diagnostic module $\mathcal{D}_1$ can also be further modified by limiting its search to recent occurrences of exogenous actions. This can be done by

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

where DM_2 is obtained by replacing an atom $0 \leq T < n$ in the bodies of rules of DM_0 by $n - m \leq T < n$. The constant m determines the time interval in the past that an agent is willing to consider in it's search for possible explanations. To simplify our discussion in the rest of the paper we *assume that* $m = 1$. Finally, the rule

$$\leftarrow k\{o(A, n-1)\}.$$

added to DM_2 will eliminate all diagnoses containing more than k actions. Of course the resulting module $\mathcal{D}_3$ as well as $\mathcal{D}_2$ can miss some diagnoses and deepening of the search and/or increase of k may be necessary if no diagnosis of a symptom is found. There are many other interesting ways of constructing efficient diagnostics modules. We are especially intrigued by the possibilities of using new features of answer sets solvers such as weight rules of SMODELS and soft constraints of DLV [19] to specify a preference relation on diagnosis. This however is a subject of further investigation. Suppose now the diagnostician has a candidate diagnosis D of a symptom $\mathcal{S}$. Is it indeed a diagnosis?

4 Finding a diagnosis

To answer this question the agent should be able to test components of $\Delta(D)$. Assuming that *no exogenous actions occur during testing* a diagnosis can be found by the following simple algorithm, $Find\_Diag(\mathcal{S})$:

function $Find\_Diag(\mathcal{S})$
  repeat
    $(E, \Delta) := Candidate\_Diag(\mathcal{S})$;

```
    diag := true;    Δ0 := Δ;
    while Δ0 ≠ ∅ and diag do
       select c ∈ Δ0;    Δ0 := Δ0 \ {c};
       if faulty(c) then
          On := On ∪ obs(ab(c), n);
       else
          On := On ∪ obs(¬ab(c), n);
          diag := false;
       end
    end {while}
 until diag or Δ = ∅;
 return (E, Δ).
```

The algorithm uses functions $Candidate\_Diag(\mathcal{S})$ which returns a candidate diagnosis (E, Δ) of $\mathcal{S}$ and $faulty(c)$ which checks if a component c of S is faulty. Notice that $\Delta = \emptyset$ indicates that no diagnosis is found - the diagnostician failed. To illustrate the algorithm, consider

Example 4. Consider the system S from Example 1 and a history Γ_0 in which b is not protected, all components of S are ok, both switches are open, and the agent closes s_1 at time 0. At time 1, he observes that the bulb b is not lit, considers $\mathcal{S} = \langle \Gamma_0, O_1 \rangle$ where $O_1 = \{obs(\neg on(b), 1)\}$ and calls function $Need\_Diag(\mathcal{S})$ which searches for an answer set of $TEST(\mathcal{S})$. There are no such sets, the diagnostician realizes he has a symptom to diagnose and calls function $Find\_Diag(\mathcal{S})$. Let us assume that the first call to $Candidate\_Diag$ returns

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

Suppose that the agent selects component r from Δ and determines that it is not faulty. Observation $obs(\neg ab(r), 1)$ will be added to O_1, *diag* will be set to $false$ and the program will call $Candidate\_Diag$ again with the updated symptom $\mathcal{S}$ as a parameter. $Candidate\_Diag$ will return another possible diagnosis

$$PD_2 = \langle \{o(brk, 0)\}, \{b\} \rangle$$

The agent will test bulb b, find it to be faulty, add observation $obs(ab(b), 1)$ to O_1 and return PD_2.

Now let us consider a different scenario:

Example 5. Let Γ_0 and observation O_1 be as in Example 4 and suppose that the program's first call to $Candidate\_Diag$ returns PD_1, b is found to be faulty, $obs(ab(b), 1)$ is added to O_1, and $Find\_Diag$ returns PD_1. The agent proceeds to have b repaired but, to his disappointment, discovers that b is still not on! Intuitively this means that PD_1 is a wrong diagnosis - there must have been a power surge at 0.

The example shows that, *in order to find a correct explanation of a symptom, it is essential for an agent to repair damaged components and observe the behavior* *of the system after repair*. For simplicity we assume that, similar to testing, repair occurs in well controlled environment, i.e. *no exogenous actions happen during the repair process*. To formally model this process we introduce a special action, $repair(c)$ for every component c of S. The effect of this action will be defined by the causal law:

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

The diagnostic process will be now modeled by the following algorithm: (Here $\mathcal{S} = \langle \Gamma_{n-1}, O \rangle$) and $\{obs(f_i, k)\}$ is a collection of observations the diagnostician makes to test his repair at moment k.)

procedure $Diagnose(\mathcal{S};$
  $k := n;$
  while $Need\_Diag(\mathcal{S})$ **do**
    $\langle E, \Delta \rangle = Find\_Diag(\mathcal{S});$
    if $\Delta = \emptyset$ **then**
      no diagnosis
    else
      $Repair(\Delta);$
      $O := O \cup \{hpd(repair(c), k) : c \in \Delta\};$
      $k := k + 1;$
      $O := O \cup \{obs(f_i, k)\};$
    end
  end

Example 6. To illustrate the above algorithm let us go back to the agent from Example 5 who just discovered diagnosis D_1. He will repair the bulb and check if the bulb is lit. It is not, and therefore a new observation is recorded as follows:

$$O_1 := O_1 \cup \{hpd(repair(b), 1), obs(\neg on(b), 2)\}$$

$Need\_Diag(\mathcal{S})$ will detect a continued need for diagnosis, $Find\_Diag(\mathcal{S})$ will return D_3, which, after new repair and testing will hopefully prove to be the right diagnosis.

The diagnosis produced by the above algorithm can be viewed as a reasonable interpretation of discrepancies between the agent's predictions and actual observations. To complete our analysis of step 1 of the agent's acting and reasoning loop we need to explain how this interpretation can be incorporated in the agent's history. If the diagnosis discovered is unique then the answer is obvious - O is simply added to Γ_{n-1}. If however faults of the system components can be caused by different sets of exogenous actions the situation becomes more subtle. Complete investigation of the issues involved is the subject of further research.

5 Related work

There is a numerous number of papers on diagnosis many of which substantially influenced the authors views on the subject. The roots of our approach go back to [15] where diagnosis for a static environment were formally defined in logical terms. Recent expansions of this work [17, 12, 3] which take into account the dynamics of system's behavior served as the starting point of the work presented in this paper. We

1. substantially simplified the basic definitions of [3];
2. presented reasonable efficient and provenly correct algorithms for computing 'dynamic' diagnosis;
3. showed how to combine diagnostics with planning and other activities of a reasoning agent.

The simplification of basic definitions from [3]is achieved by a careful choice of the 'history description' language - $\mathcal{AL}$ seems to be more suitable for our purposes that $\mathcal{L}$ used in [3]. The reasoning algorithms are based on recent discoveries of close relationship between A-Prolog and reasoning about effects of actions [11] and the ideas from answer set programming [10, 13, 9]. This approach of course would be impossible without existence of efficient answer set reasoning systems. Finally, the integration of a diagnostic and other activities is based on the agent architecture from [1].

6 Conclusion

The paper describes an ongoing work on the development of a diagnostic problem solving agent in A-Prolog. In particular we are looking for for good modeling techniques with clear and provenly correct algorithms. The following can be of interest to people who share these interests:

• definitions of a symptom, candidate diagnosis, and diagnosis which we believe to be substantially simpler than other similar approaches;

• a new algorithm for computing candidate diagnoses. (The algorithm is based on answer set programming and views the search for candidate diagnoses as 'planning in the past');

• a simple account of diagnostics, testing and repair based on the use of answer set solvers.

In the full paper we plan to give mathematical analysis of correctness of the corresponding algorithms and test them on medium size examples.

References

1. Baral, C., and Gelfond, M. Reasoning agents in dynamic domains. In Minker, J,. ed., *Logic-Based Artificial Intelligence*, Kluwer Academic Publishers, (2000), 257–279,
2. Baral, C., Gelfond, M., and Provetti, A. Reasoning about actions: laws, observations, and hypotheses. In *Journal of Logic Programming*, volume 31, 201–244, 1994.
3. Baral, C., McIlraith, S., and Son, T. Formulating diagnostic problem solving using an action language with narratives and sensing. In *Proceedings of the 2000 KR Conference*, 311-322, 2000.

4. de Kleer, J., Mackworth, A., and Reiter, R. Characterizing diagnoses and systems. In *Artificial Intelligence*, volume 56(2-3), 197–222, 1992.
5. Gelfond, M., and Lifschitz, V. The stable model semantics for logic programming. In *Logic Programming: Proc. of the Fifth Int'l Conf. and Symp.*, 1070–1080, 1988.
6. Gelfond, M., and Lifschitz, V. Classical negation in logic programs and disjunctive databases. In *New Generation Computing*, 365–387, 1991.
7. Gelfond, M., and Lifschitz, V. Representing actions in extended logic programs. In *Proc. of Joint International Conference and Symposium on Logic Programming*, 559–573, 1992.
8. Gelfond, M., and Lifschitz, V. Action languages. In *Electronic Transactions on AI*, volume 3(16), 1998.
9. Lifschitz, V. Action languages, Answer Sets, and Planning. In *The Logic Programming Paradigm: a 25-Year Perspective*. 357–373, Springer Verlag, 1999.
10. Marek, W., and Truszczynski, M. Stable models and an alternative logic paradigm. In *The Logic Programming Paradigm: a 25-Year Perspective*, 375–398, Springer Verlag, 1999.
11. McCain, T., and Turner, H. A causal theory of ramifications and qualifications. In *Artificial Intelligence*, volume 32, 57–95, 1995.
12. McIlraith, T. Explanatory diagnosis conjecturing actions to explain observations. In *Proceedings of the 1998 KR Conference*, 167–177, 1998.
13. Niemela, I. Logic programs with stable model semantics as a constraint programming paradigm. In *Annals of Mathematics and Artificial Intelligence*, 25(3-4), 241–273, 1999.
14. Niemela, I., and Simons, P. SMODELS - an implementation of the well-founded and stable model semantics for normal logic programs. In *Proc. of LPNMR'97*, volume 1265 of Lecture Notes in Computer Science, 420–429, 1997.
15. Reiter, R. A theory of diagnosis from first principles. In *Artificial Intelligence*, volume 32, 57–95, 1987.
16. Simons, P. Extending the stable model semantics with more expressive rules. In *5th International Conference, LPNMR'99*, 305–316, 1999.
17. Thielscher, M. A theory of dynamic diagnosis. In *Linkoping Electronic Articles in Computer and Information Science*, volume 2(11), 1997.
18. H. Turner. Representing actions in logic programs and default theories. In *Journal of Logic Programming*, 31(1-3):245–298, May 1997.
19. Eiter, T., Faber, Leone, N., Pfeifer, G. Declarative Problem Solving in DLV In Minker, J,. ed., *Logic-Based Artificial Intelligence*, Kluwer Academic Publishers, 257–279, 2000.