Diagnostic reasoning with A-Prolog *

MARCELLO BALDUCCINI and MICHAEL GELFOND
Department of Computer Science, Texas Tech University
Lubbock, TX 79409, USA
(*e-mail:* {mgelfond,balducci}@cs.ttu.edu)

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's components. We present simplified definitions of the notions of symptom, candidate diagnosis, and diagnosis which are based on the theory of action language $\mathcal{AL}$. The 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.

KEYWORDS: answer set programming, diagnostic reasoning, intelligent agents

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 (stable model) semantics (Gelfond and Lifschitz, 1988; Gelfond and Lifschitz, 1991) 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) satisfy the following simplifying conditions.

1. The agent's environment can be 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.
3. Normally the agent is capable of observing all relevant exogenous events occurring in its environment.

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

* This work was supported in part by United Space Alliance under Research Grant 26-3502-21 and Contract COC671311, and by NASA under Contracts 1314-44-1476 and 1314-44-1769. An extended version of this paper is available from http://www.krlab.cs.ttu.edu.

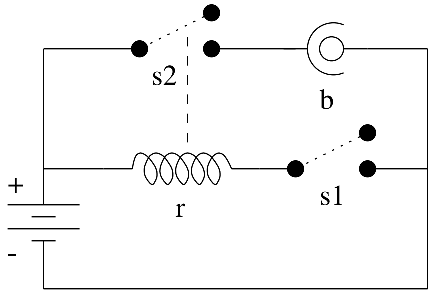

Fig. 1. $\mathcal{AC}$

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 executed in a state σ. The *second part* of the agent's knowledge, called a *recorded history* contains observations made by the agent together with a record of its own actions. It defines a collection of paths in the diagram which, from the standpoint of the agent, can be interpreted as the system's possible pasts. If the agent's knowledge is complete (e.g., 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 (the *observe-think-act-loop* (Kowalski and Sadri, 1999; Baral and Gelfond, 2000)):

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 expectations and the system's actual behavior. The following example will be used throughout the paper:

Example 1.1
Consider a system S consisting of an agent operating 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 of presentation we consider the agent capable of performing only one

action, $close(s_1)$. The environment can be represented by two damaging exogenous[1] actions: brk, which causes b to become faulty, and srg (power surge), 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 develop methods for modeling the agent's behavior after this discovery*.

We start with presenting a mathematical model of an agent and its environment based on the theory of action languages (Gelfond and Lifschitz, 1998). Even though our approach is applicable to a large collection of action languages, to simplify the discussion we will limit our attention to action language $\mathcal{AL}$ from (Baral and Gelfond, 2000). We proceed by presenting definitions of the notions of symptom, candidate diagnosis, and diagnosis which somewhat differ from those we were able to find in the literature. These definitions are used to give a simple account of the agent's behavior including diagnostics, testing, and repair. We also suggest algorithms for performing these tasks, which are based on encoding the agents knowledge in A-Prolog and reducing the agent's tasks to computing stable models (answer sets) of logic programs.

In this paper we assume that at any moment of time the agent is capable of testing whether a given component is functioning properly. Modification of the algorithms in the situation when this assumption is lifted is the subject of further research.

There is a numerous literature on automating various types of diagnostic tasks and the authors were greatly influenced by it. We mention only several papers which served as a starting point for our investigation. Of course we are indebted to R. Reiter (Reiter, 1987) which seems to contain the first clear logical account of the diagnostic problem. We were also influenced by early papers of D. Poole and K. Eshghi who related diagnostics and logic programming, seriously discussed the relationship between diagnostics and knowledge representation, and thought about the ways to combine descriptions of normal behaviour of the system with information about its faults. More recently M. Thielscher, S. McIlraith, C. Baral, T. Son, R. Otero recognized that diagnostic problem solving involves reasoning about the evolution of dynamic systems, related diagnostic reasoning with reasoning about action, change, and causation, and told the story of diagnostics which included testing and repair.

In our paper we generalize and modify this work in several directions.

- We considered a simple and powerful language $\mathcal{AL}$ for describing the agent's knowledge. Unlike some of the previous languages used for this purpose, $\mathcal{AL}$ allows concurrent actions and consecutive time-steps, and makes the distinction between observations and the derived (possibly defeasible) knowledge. The semantics of the language allows to explain malfunctioning of the system

[1] By *exogenous* actions we mean actions performed by the agent's environment. This includes natural events as well as actions performed by other agents.

by some past occurrences of exogenous (normally breaking) actions which remain unobserved by the agent.

- We simplified the basic definitions such as symptom, candidate diagnosis, and diagnosis.
- We established the realtionship between $\mathcal{AL}$ and logic programming and used this relationship to reduce various diagnostic tasks to computing stable models of logic programs.
- Finally we proved correctness of the corresponding diagnostic algorithms.

The paper is organized as follows: in Section 2 we introduce a motivating example. Section 3 introduces basic definitions used throughout the paper. In Sections 4 and 5, we show how techniques of answer set programming can be applied to the computation of candidate diagnoses and of diagnoses. In Section 6 we investigate the issues related to the introduction of the ability to repair damaged components. Section 7 discusses related work. In Section 8 we conclude the paper and describe how our work can be extended. The remaining sections contain the description of syntax and semantics of A-Prolog and $\mathcal{AL}$, as well as the proofs of the main theorems stated in this paper.

2 Modeling the domain

We start with some formal definitions describing a diagnostic domain consisting of an agent controlling a physical device. We limit ourselves to *non-intrusive* and *observable* domains in which the *agent's environment does not normally interfere with his work* and *the agent normally observes all of the domain occurrences of exogenous actions*. The agent is, however, aware of the fact that these assumptions can be contradicted by observations. As a result the agent is ready to observe and to take into account occasional occurrences of exogenous 'breaking' actions. Moreover, discrepancies between expectations and observations may force him to conclude that some exogenous actions in the past remained unobserved. This view of the relationship between the agent and his environment determined our choice of action language used for describing the agent's domain and, to the large extent, is responsible for substantial differences between our approach and that of (Baral, McIlraith, and Son, 2000).

By a *domain signature* we mean a triple $\Sigma = \langle C, F, A \rangle$ of disjoint finite sets. Elements of C will be called device *components* and used to name various parts of the device. Elements of F are referred to as *fluents* and used to denote dynamic properties of the domain [2]. By *fluent literals* we mean fluents and their negations (denoted by $\neg f$). We also assume existence of a set $F_0 \subseteq F$ which, intuitively, corresponds to the class of fluents which can be directly observed by the agent. The set of literals formed from a set $X \subseteq F$ of fluents will be denoted by $lit(X)$. A set $Y \subseteq lit(F)$ is called *complete* if for any $f \in F$, $f \in Y$ or $\neg f \in Y$; Y is called

[2] Our definitions could be easily generalized to domains with non-boolean fluents. However, the restriction to boolean fluents will simplify the presentation.

consistent if there is no f such that $f, \neg f \in Y$. We assume that for every component c the set F_0 contains a fluent $ab(c)$ which says that the device's component c is faulty. The use of ab in diagnosis goes back to (Reiter, 1987). 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*. (Occurrences of unobserved exogenous actions will be viewed as possible causes of the system's malfunctioning).

By a *transition diagram* over signature Σ we mean a directed graph T such that:

(a) the states of T are labeled by complete and consistent sets of fluent literals (corresponding to possible physical states of the domain).

(b) the arcs of T are labeled by subsets of A called *compound actions*. (Intuitively, execution of a compound action $\{a_1, \ldots, a_k\}$ corresponds to the simultaneous execution of its components).

Paths of a transition diagram correspond to *possible trajectories* of the domain. A particular trajectory, W, called the *actual trajectory* corresponds to the actual behavior of the domain. In our observe-think-act loop the agent's connection with reality is modeled by a function $observe(n, f)$ which takes a natural number n and a fluent $f \in F_0$ as parameters and returns f if f belongs to the n'th state of W and $\neg f$ otherwise

Definition 2.1
By a *diagnostic domain* we mean a triple $\langle\Sigma, T, W\rangle$ where Σ is a domain signature, T is a transition diagram over Σ, and W is the domain's actual trajectory.

To design an intelligent agent associated with a diagnostic domain $S = \langle\Sigma, T, W\rangle$ we need to supply the agent with the knowledge of Σ, T, and the recorded history of S up to a current point n. Elements of Σ can normally be defined by a simple logic program. Finding a concise and convenient way to define the transition diagram of the domain is somewhat more difficult. We start with limiting our attention to transition diagrams defined by action descriptions of action language $\mathcal{AL}$ from (Baral and Gelfond, 2000). The accurate description of the language can be found in Section 10. A typical action description SD of $\mathcal{AL}$ consists of a collection of *causal laws* determining the effects of the domain's actions, the actions' *executability conditions*, and the *state constraints* - statements describing dependences between fluents. (We often refer to statements of SD as *laws*.) 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 $\neg ab(c)$. 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. (To simplify our further discussion we only consider exogenous actions capable of causing malfunctioning of the system's components. The restriction is however inessential and can easily be lifted.)

By the *recorded history* Γ_n of S up to a current moment n we mean a collection of *observations*, i.e. statements of 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 t is an integer from the interval $[0,n)$. Notice that, intuitively, recorded history $hpd(a_1,1), hpd(a_2,1)$ says that an 'empty' action, $\{\}$, occurred at moment 0 and actions a_1 and a_2 occur concurrently at moment 1.

An agent's knowledge about the domain up to moment n will consists of an action description of $\mathcal{AL}$ and domain's recorded history. The resulting theory will often be referred to as a *domain description* of $\mathcal{AL}$.

Definition 2.2
Let S be a diagnostic domain with transition diagram T and actual trajectory $W = \langle \sigma_0^w, a_0^w, \sigma_1^w, \ldots, a_{n-1}^w, \sigma_n^w \rangle$, and let Γ_n be a recorded history of S up to moment n.
(a) A path $\langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$ in T is a *model* of Γ_n (with respect to S) if for any $0 \leq t \leq n$

1. $a_t = \{a : hpd(a,t) \in \Gamma_n\}$;
2. if $obs(l,t) \in \Gamma_n$ then $l \in \sigma_t$.

(b) Γ_n is *consistent* (with respect to S) if it has a model.

(c) Γ_n is *sound* (with respect to S) if, for any l, a, and t, if $obs(l,t), hpd(a,t) \in \Gamma_n$ then $l \in \sigma_t^w$ and $a \in a_t^w$.

(d) A fluent literal l *holds* in a model M of Γ_n at time $t \leq n$ ($M \models h(l,t)$) if $l \in \sigma_t$; Γ_n *entails* $h(l,t)$ ($\Gamma_n \models h(l,t)$) if, for every model M of Γ_n, $M \models h(l,t)$.

Notice that, in contrast to definitions from (Baral, McIlraith, and Son, 2000) based on action description language $\mathcal{L}$ from (Baral, Gelfond, and Provetti, 1994), recorded history in $\mathcal{AL}$ 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. Notice also that a recorded history may be consistent, i.e. compatible with T, but not sound, i.e. incompatible with the actual trajectory of the domain.

The following is a description, SD, of system S from Example 1.1:

$$
Objects \begin{cases} comp(r). \\ comp(b). \\ switch(s_1). \\ switch(s_2). \end{cases}
\qquad
Fluents \begin{cases} fluent(active(r)). \\ fluent(on(b)). \\ fluent(prot(b)). \\ fluent(closed(SW)) \leftarrow switch(SW). \\ fluent(ab(X)) \leftarrow comp(X). \end{cases}
$$

$$
\begin{matrix} Agent \\ Actions \end{matrix} \begin{cases} a\_act(close(s_1)). \end{cases}
\qquad
\begin{matrix} Exogenous \\ Actions \end{matrix} \begin{cases} x\_act(brk). \\ x\_act(srg). \end{cases}
$$

Causal Laws and Executability Conditions describing normal functioning of S:

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

($causes(A, L, P)$ says that execution of elementary 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 from Example 1.1 is given by:

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

Now consider a history, Γ_1 of S:

$$\Gamma_1 \begin{cases} hpd(close(s_1), 0). & obs(\neg ab(b), 0). \\ obs(\neg closed(s_1), 0). & obs(\neg ab(r), 0). \\ obs(\neg closed(s_2), 0). & obs(prot(b), 0). \end{cases}$$

Γ_1 says that, initially, the agent observed that s_1 and s_2 were open, both the bulb, b, and the relay, r, were not to be damaged, and the bulb was protected from surges. Γ_1 also contains the observation that action $close(s_1)$ occurred at time 0.

Let σ_0 be the initial state, and σ_1 be the successor state, reached by performing action $close(s_1)$ in state σ_0. It is easy to see that the path $\langle \sigma_0, close(s_1), \sigma_1 \rangle$ is the only model of Γ_1 and that $\Gamma_1 \models h(on(b), 1)$.

3 Basic definitions

Let S be a diagnostic domain with the transition diagram T, and actual trajectory $W = \langle \sigma_0^w, a_0^w, \sigma_1^w, \ldots, a_{n-1}^w, \sigma_n^w \rangle$. A pair, $\langle \Gamma_n, O_n^m \rangle$, where Γ_n is the recorded history of S up to moment n and O_n^m is a collection of observations made by the agent between times n and m, will be called a *configuration*. We say that a configuration

$$\mathcal{S} = \langle \Gamma_n, O_n^m \rangle \tag{1}$$

is a *symptom* of the system's malfunctioning if Γ_n is consistent (w.r.t. S) and $\Gamma_n \cup O_n^m$ is not. Our definition of a candidate diagnosis of symptom (1) is based on the notion of *explanation* from (Baral and Gelfond, 2000). According to that 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 $\Gamma_n \cup O_n^m \cup E$ is consistent.

Definition 3.1
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, $\Delta(D) = \{c : M \models h(ab(c), m)\}$ for some model M of $\Gamma_n \cup O_n^m \cup E(D)$.

Definition 3.2
We say that D is a *diagnosis* of a symptom $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$ if D is a candidate diagnosis of $\mathcal{S}$ in which all components in Δ are faulty, i.e., for any $c \in \Delta(D)$, $ab(c) \in \sigma_m^w$.

4 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 (Marek and Truszczynski, 1999). The proofs of the theorems presented here can be found in Section 12.

From now on, we assume that we are given a diagnostic domain $S = \langle \Sigma, T, W \rangle$. *SD* will denote an action description defining T.

Consider a system description *SD* of S whose behavior up to the moment n from some interval $[0, N)$ is described by recorded history Γ_n. (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 (Niemela and Simons, 1997). Since SMODELS takes as an input programs with finite Herbrand bases, references to lists should be eliminated from laws of *SD*. To do that we expand the signature of *SD* by new terms - names of the corresponding statements of *SD* - and consider a mapping α, from action descriptions of $\mathcal{AL}$ into programs of A-Prolog, defined as follows:

1. $\alpha(causes(a, l_0, [l_1 \dots l_m]))$ is the collection of atoms

   $$d\_law(d), head(d, l_0), action(d, a),$$
   $$prec(d, 1, l_1), \dots, prec(d, m, l_m), prec(d, m+1, nil).$$

   Here and below d will refer to the name of the corresponding law. Statement $prec(d, i, l_i)$, with $1 \leq i \leq m$, says that l_i is the i'th precondition of the law d; $prec(d, m+1, nil)$ indicates that the law has exactly m preconditions. This encoding of preconditions has a purely technical advantage. It will allow us to concisely express the statements of the form '*All preconditions of a law d are satisfied at moment T*'. (See rules (3-5) in the program Π below.)
2. $\alpha(caused(l_0, [l_1 \dots l_m]))$ is the collection of atoms

   $$s\_law(d), head(d, l_0),$$
   $$prec(d, 1, l_1), \dots, prec(d, m, l_m), prec(d, m+1, nil).$$
3. $\alpha(impossible\_if(a, [l_1 \dots l_m]))$ is a constraint

   $$\leftarrow \quad h(l_1, T), \dots, h(l_n, T), \; o(a, T).$$

where $o(a, t)$ stands for '*elementary 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:

$$\Pi \left\{ \begin{array}{lll}
1.\ h(L, T') & \leftarrow & d_law(D), \\
 & & head(D, L), \\
 & & action(D, A), \\
 & & o(A, T), \\
 & & prec_h(D, T). \\
2.\ h(L, T) & \leftarrow & s_law(D), \\
 & & head(D, L), \\
 & & prec_h(D, T). \\
3.\ all_h(D, N, T) & \leftarrow & prec(D, N, nil). \\
4.\ all_h(D, N, T) & \leftarrow & prec(D, N, P), \\
 & & h(P, T), \\
 & & all_h(D, N', T). \\
5.\ prec_h(D, T) & \leftarrow & all_h(D, 1, T). \\
6.\ h(L, T') & \leftarrow & h(L, T), \\
 & & not\ h(\overline{L}, T'). \\
7. & \leftarrow & h(L, T), h(\overline{L}, T). \\
8.\ o(A, T) & \leftarrow & hpd(A, T). \\
9.\ h(L, 0) & \leftarrow & obs(L, 0). \\
10. & \leftarrow & obs(L, T), \\
 & & not\ h(L, T).
\end{array} \right.$$

Here D, A, L are variables for the names of laws, actions, and fluent literals respectively, T, T' denote consecutive time points, and N, N' are variables for consecutive integers. (To run this program under SMODELS we need to either define the above types or add the corresponding typing predicates in the bodies of some rules of Π. These details will be omitted to save space.) The relation o is used instead of hpd to distinguish between actions observed (hpd), and actions hypothesized (o).

Relation $prec\_h(d, t)$, defined by the rule (5) of Π, says that all the preconditions of law d are satisfied at moment t. This relation is defined via an auxiliary relation $all\_h(d, i, t)$ (rules (3), (4)), which holds if the preconditions $l_i, \dots, l_m$ of d are satisfied at moment t. (Here $l_1, \dots, l_m$ stand for the ordering of preconditions of d used by the mapping α.) Rules (1),(2) of Π describe the effects of causal laws and constraints of SD. Rule (6) is the inertia axiom (McCarthy and Hayes, 1969), rule (7) rules out inconsistent states, rules (8) and (9) establish the relationship between observations and the basic relations of Π, and rule (10), called the *reality check*, guarantees that observations do not contradict the agent's expectations.

(One may be tempted to replace ternary relation $prec(D, N, P)$ by a simpler binary relation $prec(D, P)$ and to define relation $prec\_h$ by the rules:

$$\begin{array}{lcl} \neg prec\_h(D, T) & \leftarrow & prec(D, P), \neg h(P, T). \\ prec\_h(D, T) & \leftarrow & not\ \neg prec\_h(D, T). \end{array}$$

It is important to notice that this definition is incorrect since the latter rule is defeasible and may therefore conflict with the inertia axiom.)

The following terminology will be useful for describing the relationship between answer sets of $\alpha(SD, \Gamma_n)$ and models of Γ_n.

Definition 4.1
Let SD be an action description, and A be a set of literals over $lit(\alpha(SD, \Gamma_n))$. We say that A *defines* the sequence

$$\langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$$

if $\sigma_k = \{l \mid h(l, k) \in A\}$ and $a_k = \{a \mid o(a, k) \in A\}$.

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 is *complete*, i.e. for any fluent f of SD, Γ_n contains $obs(f, 0)$ or $obs(\neg f, 0)$ then M is a model of Γ_n iff M is defined by some answer set of $\alpha(SD, \Gamma_n)$.

(The theorem is similar to the result from (Turner, 1997) which deals with a different language and uses the definitions from (McCain and Turner, 1995).)

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

$$Conf(\mathcal{S}) = \alpha(SD, \Gamma_n) \cup O_n^m \cup R \tag{3}$$

where

$$R \left\{ \begin{array}{lcl} h(f, 0) & \leftarrow & not\ h(\neg f, 0). \\ h(\neg f, 0) & \leftarrow & not\ h(f, 0). \end{array} \right.$$

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 $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$ where Γ_n is consistent. Then configuration $\mathcal{S}$ is a symptom of system's malfunctioning iff program $Conf(\mathcal{S})$ 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:

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

or, in the more compact, *choice rule*, notation of SMODELS (Simons, 1999)

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

(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 $p(\overline{t})$ such that $q(\overline{t}) \in AS$. For example, program

$\{p(X) \ : \ q(X)\}.$
$q(a).$

has two answer sets: $\{q(a)\}$, and $\{p(a), q(a)\}$.)

Finding candidate diagnoses of symptom $\mathcal{S}$ can be reduced to finding answer sets of a *diagnostic program*

$$D_0(\mathcal{S}) = Conf(\mathcal{S}) \cup DM_0. \tag{4}$$

The link between answer sets and candidate diagnoses is described by the following definition.

Definition 4.2
Let SD be a system description, $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$ be a symptom of the system's malfunctioning, X be a set of ground literals, and E and δ be sets of ground atoms. We say that $\langle E, \Delta \rangle$ *is determined by* X if

$E = \{hpd(a,t) \mid o(a,t) \in X \text{ and } a \in A_e\}$, and

$\Delta = \{c \mid h(ab(c), m) \in X\}$.

Theorem 2
Let $\langle \Sigma, T, W \rangle$ be a diagnostic domain, SD be a system description of T, $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$ be a symptom of the system's malfunctioning, and E and δ be sets of ground atoms. Then,

$\langle E, \Delta \rangle$ is a candidate diagnosis of $\mathcal{S}$

iff

$\langle E, \Delta \rangle$ is determined by an answer set of $D_0(\mathcal{S})$.

The theorem justifies the following simple algorithm for computing candidate diagnosis of a symptom $\mathcal{S}$:

function *Candidate_Diag*($\mathcal{S}$: **symptom**);
  Input: a symptom $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$.
  Output: a candidate diagnosis of the symptom, or $\langle \emptyset, \emptyset \rangle$ if no candidate diagnosis could be found.
  var E : **history**;
    Δ : **set of components**;
  if $D_0(\mathcal{S})$ is consistent **then**
    select an answer set, X, of $D_0(\mathcal{S})$;
    compute $\langle E, \Delta \rangle$ determined by X;
    return $(\langle E, \Delta \rangle)$;
  else
    $E := \emptyset$; $\Delta := \emptyset$;
  return $(\emptyset, \emptyset)$.
  end

Given a symptom $\mathcal{S}$, the algorithm constructs the program $D_0(\mathcal{S})$ and passes it as an input to SMODELS (Niemela and Simons, 1997), DLV (Citrigno, Eiter, Faber, Gottlob, Koch, Leone, Mateis, Pfeifer, Scarcello, 1997), DeReS (Cholewinski, Marek, and Truszczynski, 1996), or some other answer set finder. If no answer set is found the algorithm returns $\langle \emptyset, \emptyset \rangle$. Otherwise the algorithm returns a pair $\langle E, \Delta \rangle$ extracted from some answer set X of the program. By Theorem 2 the pair is a candidate diagnosis of $\mathcal{S}$. Notice that the set E extracted from an answer set X of $D_0(\mathcal{S})$ cannot be empty and hence the answer returned by the function is unambiguos. (Indeed, using the Splitting Set Theorem (Lifschitz and Turner, 1994; Turner, 1996) we can show that the existence of answer set of $D_0(\mathcal{S})$ with empty E will lead to existence of an answer set of $Conf(\mathcal{S})$, which, by Corollary 1, contradicts to $\mathcal{S}$ being a symptom.) The algorithm can be illustrated by the following example.

Example 4.1

Let us again consider system S from Example 1.1. According to Γ_1 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 $\mathcal{S}_0 = \langle \Gamma_1, O_1 \rangle$ of malfunctioning of S. By running SMODELS on $Conf(\mathcal{S}_0)$ 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 $D_0(\mathcal{S}_0)$ 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 D_0 can be modified in many different ways. For instance, a simple modification, $D_1(\mathcal{S})$, which eliminates some candidate diagnoses containing actions unrelated to the corresponding symptom can be constructed as follows. First, let us introduce some terminology. Let $\alpha_i(SD)$ be a function that maps each impossibility condition of SD into a collection of atoms

$$imp(d), action(d, a), prec(d, m+1, nil), prec(d, 1, l_1), \ldots, prec(d, m, l_m),$$

where d is a new constant naming the condition, and a, l_i's are arguments of the condition. Let also REL be the following program:

$$REL \left\{ \begin{array}{llll}
1. & rel(A, L) & \leftarrow & d_law(D), \\
 & & & head(D, L), \\
 & & & action(D, A). \\
2. & rel(A, L) & \leftarrow & law(D), \\
 & & & head(D, L), \\
 & & & prec(D, N, P), \\
 & & & rel(A, P). \\
3. & rel(A_2, L) & \leftarrow & rel(A_1, L), \\
 & & & imp(D), \\
 & & & action(D, A_1), \\
 & & & prec(D, N, P), \\
 & & & rel(A_2, \overline{P}). \\
4. & rel(A) & \leftarrow & obs(L, T), \\
 & & & T \geq n, \\
 & & & rel(A, L). \\
5. & & \leftarrow & T < n, \\
 & & & o(A, T), \\
 & & & x_act(A), \\
 & & & not\ hpd(A, T), \\
 & & & not\ rel(A).
\end{array} \right.$$

and

$$DM_1 = DM_0 \cup REL \cup \alpha_i(SD).$$

The new diagnostic module, D_1 is defined as

$$D_1(\mathcal{S}) = Conf(\mathcal{S}) \cup DM_1.$$

(It is easy to see that this modification is *safe*, i.e. D_1 will not miss any useful predictions about the malfunctioning components.) The difference between $D_0(\mathcal{S})$ and $D_1(\mathcal{S})$ can be seen from the following example.

Example 4.2
Let us expand the system S from Example 1.1 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 1.1 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 4.1 together with new diagnoses containing a and $ab(c)$, e.g.

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

Diagnostic module 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 exogenous actions, including those unrelated to observations, to actually happen in the past. Constraint (5) of program *REL* only prohibits generating such actions in our search for diagnosis.

There are many other ways of improving quality of candidate diagnoses by eliminating some redundant or unlikely diagnoses, and by ordering the corresponding search space. For instance, even more unrelated actions can be eliminated from the search space of our diagnostic modules by considering relevance relation *rel* depending on time. This can be done by a simple modification of program REL which is left as an exercise to the reader. The diagnostic module D_1 can also be further modified by limiting its search to recent occurrences of exogenous actions. This can be done by

$$D_2(\mathcal{S}) = Conf(\mathcal{S}) \cup DM_2$$

where DM_2 is obtained by replacing 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 its search for possible explanations. To simplify our discussion in the rest of the paper we *assume that* $m = 1$. Finally, the rule

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

added to DM_2 will eliminate all diagnoses containing more than k actions. Of course the resulting module D_3 as well as D_2 can miss some candidate 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 and minimize of SMODELS and weak constraints of DLV (Citrigno, Eiter, Faber, Gottlob, Koch, Leone, Mateis, Pfeifer, Scarcello, 1997; Buccafurri, Leone, and Rullo, 1997) to specify a preference relation on diagnoses. This however is a subject of further investigation.

5 Finding a diagnosis

Suppose now the diagnostician has a candidate diagnosis D of a symptom $\mathcal{S}$. Is it indeed 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 $Find\_Diag($ **var** $\mathcal{S}$: **symptom**);
  Input: a symptom $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$.

Output: a diagnosis of the symptom, or $\langle \emptyset, \emptyset \rangle$ if no diagnosis
could be found. Upon successful termination of the loop, the set O_n^m
is updated in order to incorporate the results of the tests
done during the search for a diagnosis.

var O, E : **history**;
  Δ, Δ_0 : **set of components**;
  $diag$: **bool**;
$O := O_n^m$;
repeat
  $\langle E, \Delta \rangle := Candidate\_Diag(\ \langle \Gamma_n, O \rangle\ )$;
  if $E = \emptyset$ { no diagnosis could be found }
    return($\langle E, \Delta \rangle$);
  diag := $true$; $\Delta_0 := \Delta$;
  while $\Delta_0 \neq \emptyset$ **and** diag **do**
    select $c \in \Delta_0$; $\Delta_0 := \Delta_0 \setminus \{c\}$;
    if $observe(m, ab(c)) = ab(c)$ **then**
      $O := O \cup obs(ab(c), m)$;
    else
      $O := O \cup obs(\neg ab(c), m)$;
      diag := $false$;
    end
  end {while}
until diag;
$O_n^m := O$;
return ($\langle E, \Delta \rangle$).

The properties of *Find_Diag* are described by the following theorem.

Theorem 3
Let $\langle \Sigma, T, W \rangle$ be a diagnostic domain, SD be a system description of T, and $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$ be a symptom of the system's malfunctioning. Then,

1. $Find\_Diag(\mathcal{S})$ terminates;
2. let $\langle E, \Delta \rangle = Find\_Diag(\mathcal{S})$, where the value of variable $\mathcal{S}$ is set to $\mathcal{S}_0$. If $\Delta \neq \emptyset$, then

   $\langle E, \Delta \rangle$ is a diagnosis of $\mathcal{S}_0$;

   otherwise, $\mathcal{S}_0$ has no diagnosis.

To illustrate the algorithm, consider the following example.

Example 5.1
Consider the system S from Example 1.1 and a history Γ_1 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 $\mathcal{S} = \langle \Gamma_1, O_1 \rangle$ where $O_1 = \{obs(\neg on(b), 1)\}$ and calls function $Need\_Diag(\mathcal{S})$ which searches for an answer set of $Conf(\mathcal{S})$. 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. If, however, according to our actual trajectory, W, the bulb is still ok, the function returns $\langle \emptyset, \emptyset \rangle$. No diagnosis is found and the agent (or its designers) should start looking for a modeling error.

6 Diagnostics and repair

Now let us consider a scenario which is only slightly different from that of the previous example.

Example 6.1
Let Γ_1 and observation O_1 be as in Example 5.1 and suppose that the program's first call to *Candidate_Diag* returns PD_2, b is found to be faulty, $obs(ab(b), 1)$ is added to O_1, and *Find_Diag* returns PD_2. The agent proceeds to have b repaired but, to his disappointment, discovers that b is still not on! Intuitively this means that PD_2 is a wrong diagnosis - there must have been a power surge at 0.

For simplicity we assume that, similar to testing, repair occurs in well controlled environment, i.e. *no exogenous actions happen during the repair process*. 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*. 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 $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$ and $\{obs(f_i, k)\}$ is a collection of observations the diagnostician makes to test his repair at moment k.)

function *Diagnose*($\mathcal{S}$) : **boolean**;
  Input: a symptom $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$.
  Output: *false* if no diagnosis can be found. Otherwise
    repairs the system, updates O_n^m, and returns *true*.
  var E : **history**;
    Δ : **set of components**;
  $E = \emptyset$;

while $Need\_Diag(\langle \Gamma_n \cup E, O_n^m \rangle)$ **do**
  $\langle E, \Delta \rangle = Find\_Diag(\langle \Gamma_n, O_n^m \rangle)$;
  if $E = \emptyset$ **then return**(false)
  else
    $Repair(\Delta)$;
    $O_n^m := O_n^m \cup \{hpd(repair(c), m) : c \in \Delta\}$;
    $m := m + 1$;
    $O_n^m := O_n^{m-1} \cup \{obs(f_i, m)\}$;
  end
end
return(true);

Example 6.2
To illustrate the above algorithm let us go back to the agent from Example 6.1 who just discovered diagnosis $PD_2 = \langle \{o(brk, 0)\}, \{b\} \rangle$. 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 PD_1, 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. 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.

7 Related work

There is a numerous collection of papers on diagnosis many of which substantially influenced the author's views on the subject. The roots of our approach go back to (Reiter, 1987) where diagnosis for a static environment were formally defined in logical terms. To the best of our knowledge the first published extensions of this work to dynamic domains appeared in (Thielscher, 1997b), where dynamic domains were described in fluent calculus (Thielscher, 1998), and in (McIlraith, 1997) which used situation calculus (McCarthy and Hayes, 1969). Explanation of malfunctioning of system components in terms of unobserved exogenous actions was first clearly articulated in (McIlraith, 1998). Generalization and extensions of these ideas (Baral, McIlraith, and Son, 2000) which specifies dynamic domains in action language $\mathcal{L}$, can be viewed as a starting point of the work presented in this paper. The use of a simpler action language $\mathcal{AL}$ allowed us to substantially simplify the basic definitions of (Baral, McIlraith, and Son, 2000) and to reduce the computation of diagnosis to finding stable models of logic programs. As a result we were able to incorporate diagnostic reasoning in a general agent architecture based on the answer set programming paradigm, and to combine diagnostics with planning and other activities of a reasoning agent. On another hand (Baral, McIlraith, and Son, 2000) addresses some questions which are not fully addressed by our paper. In particular, the underlying action language of (Baral, McIlraith, and Son, 2000) allows non-deterministic and knowledge-producing actions absent in our work. While our formulation allows immediate incorporation of the former, incorporation of the latter seems to substantially increase conceptual complexity of the formalism. This is of course the case in (Baral, McIlraith, and Son, 2000) too but we believe that the need for such increase in complexity remains an open question. Another interesting related work is (Otero and Otero, 2000). In this paper the authors address the problem of dynamic diagnosis using the notion of pertinence logic from (Otero and Cabalar, 1999). The formalism allows to define dynamic diagnosis which, among other things, can model intermittent faults of the system. As a result it provides a logical account of the following scenario: Consider a person trying to shoot a turkey. Suppose that the gun is initially loaded, the agent shoots, observes that the turkey is not dead, and shoots one more time. Now the turkey is dead. The pertinence formalism of (Otero and Otero, 2000) does not claim inconsistency - it properly determines that the gun has an intermittent fault. Our formalism on another hand is not capable of modeling this scenario - to do that we need to introduce non-deterministic actions. Since, in our opinion, the use of pertinence logic substantially complicates action formalisms it is interesting to see if such use for reasoning with intermittent faults can always be avoided by introducing non-determinism. Additional comparison of the action languages based approach to diagnosis with other related approaches can be found in (Baral, McIlraith, and Son, 2000).

Finally, let us mention that the reasoning algorithms proposed in this paper are based on recent discoveries of close relationship between A-Prolog and reasoning about effects of actions (McCain and Turner, 1995) and the ideas from answer set programming (Marek and Truszczynski, 1999; Niemela, 1999; Lifschitz, 1999). This approach of course would be impossible without existence of efficient answer set reasoning systems. The integration of diagnostics and other activities is based on the agent architecture from (Baral and Gelfond, 2000).

8 Conclusions and further work

The paper describes a work on the development of a diagnostic problem solving agent in which a mathematical model of an agent and its environment is based on the theory of action language $\mathcal{AL}$ from (Baral and Gelfond, 2000). The language, which contains the means for representing concurrent actions and fairly complex relations between fluents, is used to give concise descriptions of transition diagrams characterizing possible trajectories of the agent domains as well as the domains' recorded histories. In this paper we:

- Establish a close relationship between $\mathcal{AL}$ and logic programming under the

answer set semantics which allows reformulation of the agent's knowledge in A-Prolog. These results build on previous work connecting action languages and logic programming.
- Give definitions of symptom, candidate diagnosis, and diagnosis which we believe to be simpler than similar definitions we were able to find in the literature.
- Suggest 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'.)
- Suggest some simple ways of using A-Prolog to declaratively limit the diagnostician's search space. This leads to higher quality diagnosis and substantial improvements in the diagnostician's efficiency.
- Give a simple account of diagnostics, testing and repair based on the use of answer set solvers. The resulting algorithms, which are shown to be provenly correct, can be easily incorporated in the agent's architecture from (Baral and Gelfond, 2000).

In our further work we plan to:

- Expand our results to more expressive languages, i.e. those with non-deterministic actions, defeasible causal laws, etc.
- Find more powerful declarative ways of limiting the diagnostician's search space. This can be done by expanding A-Prolog by ways of expressing preferences between different rules or by having the agent plan observations aimed at eliminating large clusters of possible diagnosis. In investigating these options we plan to build on related work in (Buccafurri, Leone, and Rullo, 1997) and (Baral, McIlraith, and Son, 2000; McIlraith and Scherl, 2000).
- Test the efficiency of the suggested algorithm on medium size applications.

9 The syntax and semantics of A-Prolog

In this section we give a brief introduction to the syntax and semantics of a comparatively simple variant of A-Prolog. The syntax of the language is determined by a signature Σ consisting of types, $types(\Sigma) = \{\tau_0, \dots, \tau_m\}$, object constants $obj(\tau, \Sigma) = \{c_0, \dots, c_m\}$ for each type τ, and typed function and predicate constants $func(\Sigma) = \{f_0, \dots, f_k\}$ and $pred(\Sigma) = \{p_0, \dots, p_n\}$. We will assume that the signature contains symbols for integers and for the standard relations of arithmetic. Terms are built as in typed first-order languages; positive literals (or atoms) have the form $p(t_1, \dots, t_n)$, where t's are terms of proper types and p is a predicate symbol of arity n; negative literals are of the form $\neg p(t_1, \dots, t_n)$. In our further discussion we often write $p(t_1, \dots, t_n)$ as $p(\overline{t})$. The symbol $\neg$ is called *classical* or *strong* negation. Literals of the form $p(\overline{t})$ and $\neg p(\overline{t})$ are called contrary. By $\overline{l}$ we denote a literal contrary to l. Literals and terms not containing variables are called *ground*. The sets of all ground terms, atoms and literals over Σ will be denoted by $terms(\Sigma)$, $atoms(\Sigma)$ and $lit(\Sigma)$ respectively. For a set P of predicate symbols from Σ, $atoms(P, \Sigma)$ ($lit(P, \Sigma)$) will denote the sets of ground atoms (literals) of

Σ formed with predicate symbols from P. Consistent sets of ground literals over signature Σ, containing all arithmetic literals which are true under the standard interpretation of their symbols, are called *states* of Σ and denoted by $states(\Sigma)$.

A rule of A-Prolog is an expression of the form

$$l_0 \leftarrow l_1, \ldots, l_m, not\ l_{m+1}, \ldots, not\ l_n \tag{5}$$

where $n \geq 1$, l_i's are literals, l_0 is a literal or the symbol $\bot$, and *not* is a logical connective called *negation as failure* or *default negation*. An expression *not* l says that there is no reason to believe in l. An *extended literal* is an expression of the form l or *not* l where l is a literal. A rule (5) is called a *constraint* if $l_0 = \bot$.

Unless otherwise stated, we assume that the $l's$ in rules (5) are ground. Rules with variables (denoted by capital letters) will be used only as a shorthand for the sets of their ground instantiations. This approach is justified for the so called closed domains, i.e. domains satisfying the domain closure assumption (Reiter, 1978) which asserts that *all objects in the domain of discourse have names in the language of* Π.

A pair $\langle \Sigma, \Pi \rangle$ where Σ is a signature and Π is a collection of rules over Σ is called a *logic program*. (We often denote such pair by its second element Π. The corresponding signature will be denoted by $\Sigma(\Pi)$.)

We say that a literal $l \in lit(\Sigma)$ is *true* in a state X of Σ if $l \in X$; l is *false* in X if $\overline{l} \in X$; Otherwise, l is unknown. $\bot$ is false in X.

Given a signature Σ and a set of predicate symbols E, $lit(\Sigma, E)$ denotes the set of all literals of Σ formed by predicate symbols from E. If Π is a ground program, $lit(\Pi)$ denotes the set of all atoms occurring in Π, together with their negations, and $lit(\Pi, E)$ denotes the set of all literals occurring in $lit(\Pi)$ formed by predicate symbols from E.

The answer set semantics of a logic program Π assigns to Π a collection of *answer sets* – consistent sets of ground literals over signature $\Sigma(\Pi)$ corresponding to beliefs which can be built by a rational reasoner on the basis of rules of Π. In the construction of these beliefs the reasoner is assumed to be guided by the following informal principles:

- He should satisfy the rules of Π, understood as constraints of the form: *If one believes in the body of a rule one must belief in its head.*
- He cannot believe in $\bot$ (which is understood as falsity).
- He should adhere to the *rationality principle* which says that *one shall not believe anything he is not forced to believe.*

The precise definition of answer sets will be first given for programs whose rules do not contain default negation. Let Π be such a program and let X be a state of $\Sigma(\Pi)$. We say that X is *closed* under Π if, for every rule $head \leftarrow body$ of Π, head is true in X whenever *body* is true in X. (For a constraint this condition means that the body is not contained in X.)

Definition 9.1
(Answer set – part one)
A state X of $\Sigma(\Pi)$ is an *answer set* for Π if X is minimal (in the sense of set-theoretic inclusion) among the sets closed under Π.

It is clear that a program without default negation can have at most one answer set. To extend this definition to arbitrary programs, take any program Π, and let X be a state of $\Sigma(\Pi)$. The *reduct*, Π^X, of Π relative to X is the set of rules

$$l_0 \leftarrow l_1, \ldots, l_m$$

for all rules (5) in Π such that $l_{m+1}, \ldots, l_n \notin X$. Thus Π^X is a program without default negation.

Definition 9.2
(Answer set – part two)
A state X of $\Sigma(\Pi)$ is an answer set for Π if X is an answer set for Π^X.

(The above definition differs slightly from the original definition in (Gelfond and Lifschitz, 1991), which allowed the inconsistent answer set, $lit(\Sigma)$. Answer sets defined in this paper correspond to consistent answer sets of the original version.)

10 Syntax and semantics of the causal laws of $\mathcal{AL}$

An action description of $\mathcal{AL}$ is a collection of propositions of the form

1. $causes(a_e, l_0, [l_1, \ldots, l_n])$,
2. $caused(l_0, [l_1, \ldots, l_n])$, and
3. $impossible\_if(a_e, [l_1, \ldots, l_n])$

where a_e is an elementary action and $l_0, \ldots, l_n$ are fluent literals from Σ. The first proposition says that, if the elementary action a_e were to be executed in a situation in which $l_1, \ldots, l_n$ hold, the fluent literal l_0 will be caused to hold in the resulting situation. Such propositions are called *dynamic causal laws*. The second proposition, called a *static causal law*, says that, in an arbitrary situation, the truth of fluent literals, $l_1, \ldots, l_n$ is sufficient to cause the truth of l_0. The last proposition says that action a_e cannot be performed in any situation in which $l_1, \ldots, l_n$ hold. (The one presented here is actually a simplification of $\mathcal{AL}$. Originally *impossible_if* took as argument a compound action rather than an elementary one. The restriction on a_e being elementary is not essential and can be lifted. We require it to simplify the presentation). To define the transition diagram, T, given by an action description $\mathcal{A}$ of $\mathcal{AL}$ we use the following terminology and notation. A set S of fluent literals is closed under a set Z of static causal laws if S includes the head, l_0, of every static causal law such that $\{l_1, \ldots, l_n\} \subseteq S$. The set $Cn_Z(S)$ of *consequences* of S under Z is the smallest set of fluent literals that contains S and is closed under Z. $E(a_e, \sigma)$ stands for the set of all fluent literals l_0 for which there is a dynamic causal law $causes(a_e, l_0, [l_1, \ldots, l_n])$ in $\mathcal{A}$ such that $[l_1, \ldots, l_n] \subseteq \sigma$. $E(a, \sigma) = \bigcup_{a_e \in a} E(a_e, \sigma)$. The transition system $T = \langle \mathcal{S}, \mathcal{R} \rangle$ *described* by an action description $\mathcal{A}$ is defined as follows:

1. $\mathcal{S}$ is the collection of all complete and consistent sets of fluent literals of Σ closed under the static laws of $\mathcal{A}$,
2. $\mathcal{R}$ is the set of all triples $\langle \sigma, a, \sigma' \rangle$ such that $\mathcal{A}$ does not contain a proposition of the form *impossible_if* $(a, [l_1, \ldots, l_n])$ such that $[l_1, \ldots, l_n] \subseteq \sigma$ and

$$\sigma' = Cn_Z(E(a, \sigma) \cup (\sigma \cap \sigma')) \tag{6}$$

where Z is the set of all static causal laws of $\mathcal{A}$. The argument of $Cn(Z)$ in (6) is the union of the set $E(a, \sigma)$ of the "direct effects" of a with the set $\sigma \cap \sigma'$ of facts that are "preserved by inertia". The application of $Cn(Z)$ adds the "indirect effects" to this union.

We call an action description *deterministic* if for any state σ_0 and action a there is at most one such successor state σ_1.

The above definition of T is from (McCain and Turner, 1997) and is the product of a long investigation of the nature of causality. (See for instance, (Lifschitz, 1997; Thielscher, 1997a).) Finding this definition required the good understanding of the nature of causal effects of actions in the presence of complex interrelations between fluents. An additional level of complexity is added by the need to specify what is not changed by actions. The latter, known as the *frame problem*, is often reduced to the problem of finding a concise and accurate representation of the inertia axiom – a default which says that *things normally stay as they are* (McCarthy and Hayes, 1969). The search for such a representation substantially influenced AI research during the last twenty years. An interesting account of history of this research together with some possible solutions can be found in (Shanahan, 1997).

11 Properties of logic programs

In this section we introduce several properties of logic programs which will be used, in the next appendix, to prove the main theorem of this paper.

We begin by summarizing two useful definitions from (Brass and Dix, 1994).

Definition 11.1
Let q be a literal and P be a logic program. The *definition* of q in P is the set of all rules in P which have q as their head.

Definition 11.2 (*Partial Evaluation*)
Let q be a literal and P be a logic program. Let

$$\begin{array}{lll} q & \leftarrow & \Gamma_1. \\ q & \leftarrow & \Gamma_2. \\ & \ldots & \end{array}$$

be the definition of q in P. The Partial Evaluation of P w.r.t. q (denoted by $e(P, q)$) is the program obtained from P by replacing every rule of the form

$$p \leftarrow \Delta_1, q, \Delta_2.$$

with rules

$$
\begin{array}{lcl}
p & \leftarrow & \Delta_1, \Gamma_1, \Delta_2. \\
p & \leftarrow & \Delta_1, \Gamma_2, \Delta_2. \\
 & \dots &
\end{array}
$$

Notice that, according to Brass-Dix Lemma (Brass and Dix, 1994), P and $e(P, q)$ are equivalent (written $P \simeq e(P, q)$), i.e. they have the same answer sets.

The following expands on the results from (Brass and Dix, 1994).

Definition 11.3 (*Extended Partial Evaluation*)
Let P be a ground program, and $\vec{q} = \langle q_1, q_2, \dots, q_n \rangle$ be a sequence of literals. The Extended Partial Evaluation of P w.r.t. $\vec{q}$ (denoted by $e(P, \vec{q})$) is defined as follows:

$$
e(P, \vec{q}) = \begin{cases} P & \text{if } n = 0 \\ e(e(P, q_n), \langle q_1, q_2, \dots, q_{n-1} \rangle) & \text{otherwise} \end{cases} \tag{7}
$$

From now on, the number of elements of $\vec{q}$ will be denoted by $|\vec{q}|$.

Definition 11.4 (*Trimming*)
Let $\vec{q}$ and P be as above. The Trimming of P w.r.t. $\vec{q}$ (denoted by $t(P, \vec{q})$) is the program obtained by dropping the definition of the literals in $\vec{q}$ from $e(P, \vec{q})$.

Lemma 1
Let P and R be logic programs, such that

$$
P \simeq R. \tag{8}
$$

Then, for any sequence of literals $\vec{q}$,

$$
e(P, \vec{q}) \simeq e(R, \vec{q}). \tag{9}
$$

Lemma 2
Let $\vec{q}$ be a sequence of literals and P be a logic program. Then,

$$
P \simeq e(P, \vec{q}). \tag{10}
$$

The following expands similar results from (Gelfond and Son, 1998), making them suitable for our purposes.

Definition 11.5 (*Strong Conservative Extension*)
Let P_1 and P_2 be ground programs such that $lit(P_1) \subseteq lit(P_2)$. Let Q be $lit(P_2) \setminus lit(P_1)$.

We say that P_2 is a Strong Conservative Extension of P_1 w.r.t. Q (and write $P_2 \succ_Q P_1$) if:

- if A is an answer set of P_2, $A \setminus Q$ is an answer set of P_1;
- if A is an answer set of P_1, there exists a subset B of Q such that $A \cup B$ is an answer set of P_2.

Lemma 3
Let P be a ground program, $Q \subseteq lit(P)$, and $\vec{q} = \langle q_1, \ldots, q_n \rangle$ be an ordering of Q. If $Q \cap lit(t(P, \vec{q})) = \emptyset$, then

$$P \text{ is a Strong Conservative Extension of } t(P, \vec{q}) \text{ w.r.t } Q\cdot \tag{11}$$

12 Proofs of the theorems

12.1 Proof of Theorem 1

The proof of Theorem 1 will be given in several steps.

First of all, we define a simplified encoding of an action description, SD, in A-Prolog. Then, we prove that the answer sets of the programs generated using this encoding correspond exactly to the paths in the transition diagram described by SD.

Later, we extend the new encoding and prove that, for every recorded history Γ_n, the models of Γ_n are in a one-to-one correspondence with the answer sets of the programs generated by this second encoding.

Finally, we prove that programs obtained by applying this second encoding are essentially 'equivalent' to those generated with the encoding presented in Section 4, which completes the proof of Theorem 1. In addition, we present a corollary that extends the theorem to the case in which the initial situation of Γ_n is not complete.

12.1.1 Step 1

The following notation will be useful in our further discussion. Given a time point t, a state σ, and a compound action a, let

$$\begin{array}{rcl} h(\sigma, t) & = & \{h(l, t) \mid l \in \sigma\} \\ o(a, t) & = & \{o(a', t) \mid a' \in a\}. \end{array} \tag{12}$$

These sets can be viewed as the representation of σ and a in A-Prolog.

Definition 12.1
Let SD be an action description of $\mathcal{AL}$, n be a positive integer, and $\Sigma(SD)$ be the signature of SD. $\Sigma_d^n(SD)$ denotes the signature obtained as follows:

- $const(\Sigma_d^n(SD)) = \langle const(\Sigma(SD)) \cup \{0, \ldots, n\} \rangle$;
- $pred(\Sigma_d^n(SD)) = \{h, o\}$.

Let

$$\alpha_d^n(SD) = \langle \Pi_d^\alpha(SD), \Sigma_d^n(SD) \rangle, \tag{13}$$

where

$$\Pi_d^\alpha(SD) = \bigcup_{r \in SD} \alpha_d(r), \tag{14}$$

and $\alpha_d(r)$ is defined as follows:

- $\alpha_d(causes(a, l_0, [l_1, \ldots, l_m]))$ is

$$h(l_0, T') \leftarrow h(l_1, T), \ldots, h(l_m, T), o(a, T).$$

- $\alpha_d(caused(l_0, [l_1, \ldots, l_m]))$ is

$$h(l_0, T) \leftarrow h(l_1, T), \ldots, h(l_m, T). \tag{15}$$

- $\alpha_d(impossible\_if(a, [l_1, \ldots, l_m]))$ is

$$\begin{array}{ll} \leftarrow & h(l_1, T), \ldots, h(l_m, T), \\ & o(a, T). \end{array}$$

Let also

$$\beta_d^n(SD) = \langle \Pi_d^\beta(SD), \Sigma_d^n(SD) \rangle, \tag{16}$$

where

$$\Pi_d^\beta(SD) = \Pi_d^\alpha(SD) \cup \Pi_d \tag{17}$$

and Π_d is the following set of rules:

$$\begin{array}{llll} 1. & h(L, T') & \leftarrow & h(L, T), \\ & & & \text{not } h(\overline{L}, T'). \\ 2. & & \leftarrow & h(L, T), h(\overline{L}, T). \end{array}$$

When we refer to a single action description, we will often drop the argument from $\Sigma_d^n(SD), \alpha_d^n(SD), \Pi_d^\alpha(SD), \beta_d^n(SD), \Pi_d^\beta(SD)$ in order to simplify the presentation.

For the rest of this section, we will restrict attention to ground programs. In order to keep notation simple, we will use α_d^n, β_d^n, α^n and β^n to denote the ground versions of the programs previously defined.

For any action description SD, state σ_0 and action a_0, let $\beta_d^n(SD, \sigma_0, a_0)$ denote

$$\beta_d^n \cup h(\sigma_0, 0) \cup o(a_0, 0). \tag{18}$$

We will sometimes drop the first argument, and denote the program by $\beta_d^n(\sigma_0, a_0)$.

The following lemma will be helpful in proving the main result of this subsection. It states the correspondence between (single) transitions of the transition diagram and answer sets of the corresponding A-Prolog program.

Lemma 4
Let SD be an action description, $\mathcal{T}(SD)$ be the transition diagram it describes, and $\beta_d^1(\sigma_0, a_0)$ be defined as in (18). Then, $\langle \sigma_0, a_0, \sigma_1 \rangle \in \mathcal{T}(SD)$ iff $\sigma_1 = \{l \mid h(l, 1) \in A\}$ for some answer set A of $\beta_d^1(\sigma_0, a_0)$.

Proof
Let us define

$$I = h(\sigma_0, 0) \cup o(a_0, 0) \quad (19)$$

and

$$\beta_d^1 = \beta_d^1(\sigma_0, a_0) \cup I\cdot$$

Left-to-right. Let us show that, if $\langle \sigma_0, a_0, \sigma_1 \rangle \in \mathcal{T}(SD)$,

$$A = I \cup h(\sigma_1, 1) \quad (20)$$

is an answer set of $\beta_d^1(\sigma_0, a_0)$. Notice that $\langle \sigma_0, a_0, \sigma_1 \rangle \in \mathcal{T}(SD)$ implies that σ_1 is a state.

Let us prove that A is the minimal set of atoms closed under the rules of the reduct P^A. P^A contains:

a) set I;
b) all rules in $\alpha_d^1(SD)$ (see (13));
c) a constraint $\leftarrow h(l, t), h(\overline{l}, t).$ for any fluent literal l and time point t;
d) a rule

$$h(l, 1) \leftarrow h(l, 0)$$

for every fluent literal l such that $h(l, 1) \in A$ (in fact, since σ_1 is complete and consistent, $h(l, 1) \in A \Leftrightarrow h(\overline{l}, 1) \notin A$).

<u>A is closed under P^A</u>. We will prove it for every rule of the program.

Rules of groups (a) and (d): obvious.

Rules of group (b) encoding dynamic laws of the form $causes(a, l, [l_1, \ldots, l_m])$:

$$h(l, 1) \quad \leftarrow \quad h(l_1, 0), \ldots, h(l_m, 0), o(a, 0).$$

If $\{h(l_1, 0), \ldots, h(l_m, 0), o(a, 0)\} \subseteq A$, then, by (20), $\{l_1, \ldots, l_m\} \subseteq \sigma_0$ and $a \in a_0$. Therefore, the preconditions of the dynamic law are satisfied by σ_0. Hence (6) implies $l \in \sigma_1$. By (20), $h(l, 1) \in A$.

Rules of group (b) encoding static laws of the form $caused(l, [l_1, \ldots, l_m])$:

$$h(l, t) \quad \leftarrow \quad h(l_1, t), \ldots, h(l_m, t).$$

If $\{h(l_1, t), \ldots, h(l_m, t)\} \subseteq A$, then, by (20), $\{l_1, \ldots, l_m\} \subseteq \sigma_t$, i.e. the preconditions of the static law are satisfied by σ_t. If $t = 1$, then (6) implies $l \in \sigma_1$. By (20), $h(l, t) \in A$. If $t = 0$, since states are closed under the static laws of SD, we have that $l \in \sigma_0$. Again by (20), $h(l, t) \in A$.

Rules of group (b) encoding impossibility laws of the form $impossible\_if(a, [l_1, \ldots, l_m])$:

$$\leftarrow \quad h(l_1, 0), \ldots, h(l_m, 0), o(a, 0).$$

Since $\langle \sigma_0, a_0, \sigma_1 \rangle \in \mathcal{T}(SD)$ by hypothesis, $\langle \sigma_0, a_0 \rangle$ does not satisfy the preconditions of any impossibility condition. Then, either $a \notin a_0$ or $l_i \notin \sigma_0$ for some i. By (20), the body of this rule is not satisfied.

Rules of group (c). Since σ_0 and σ_1 are consistent by hypothesis, l and $\overline{l}$ cannot both belong to the same state. By (20), either $h(l,0) \notin A$ or $h(\overline{l},0) \notin A$, and the same holds for time point 1. Therefore, the body of these rules is never satisfied.

<u>A is the minimal set closed under the rules of P^A</u>. We will prove this by assuming that there exists a set $B \subseteq A$ such that B is closed under the rules of P^A, and by showing that $B = A$.

First of all,

$$I \subseteq B, \tag{21}$$

since these are facts in P^A.

Let

$$\delta = \{l \mid h(l,1) \in B\}. \tag{22}$$

Since $B \subseteq A$,

$$\delta \subseteq \sigma_1 \tag{23}$$

We will show that $\delta = \sigma_1$ by proving that

$$\delta = CN_Z(E(a_0, \sigma_0) \cup (\sigma_1 \cap \sigma_0)). \tag{24}$$

<u>Dynamic laws</u>. Let d be a dynamic law of *SD* of the form $causes(a, l_0, [l_1, \ldots, l_m])$, such that $a \in a_0$ and $\{l_1, \ldots, l_m\} \subseteq \sigma_0$. Because of (21), $h(\{l_1, \ldots, l_m\}, 0) \subseteq B$ and $o(a,0) \in B$. Since B is closed under $\alpha_d(d)$ (15), $h(l_0, 1) \in B$, and $l_0 \in \delta$. Therefore, $E(a_0, \sigma_0) \subseteq \delta$.

<u>Inertia</u>. P^A contains a (reduced) inertia rule of the form

$$h(l,1) \leftarrow h(l,0). \tag{25}$$

for every literal $l \in \sigma_1$. Suppose $l \in \sigma_1 \cap \sigma_0$. Then, $h(l,0) \in I$, and, since B is closed under (25), $h(l,1) \in B$. Therefore, $\sigma_1 \cap \sigma_0 \subseteq \delta$.

<u>Static laws</u>. Let s be a static law of *SD* of the form $caused(l_0, [l_1, \ldots, l_m])$, such that

$$h(\{l_1, \ldots, l_m\}, 0) \subseteq B. \tag{26}$$

Since B is closed under $\alpha_d(s)$ (15), $h(l_0, 1) \in B$, and $l_0 \in \delta$. Then, δ is closed under the static laws of *SD*.

Summing up, (24) holds. From (6) and (23), we obtain $\sigma_1 = \delta$. Therefore $h(\sigma_1, 1) \subseteq B$.

At this point we have shown that $I \cup h(\sigma_1, 1) \subseteq B \subseteq A$.

Right-to-left. Let A be an answer set of P such that $\sigma_1 = \{l \mid h(l,1) \in A\}$. We have to show that

$$\sigma_1 = CN_Z(E(a_0, \sigma_0) \cup (\sigma_1 \cap \sigma_0)), \tag{27}$$

that $\langle \sigma_0, a_0 \rangle$ respects all impossibility conditions, and that σ_1 is consistent and complete.

$\underline{\sigma_1 \text{ consistent}}$. Obvious, since A is a (consistent) answer set by hypothesis.

$\underline{\sigma_1 \text{ complete}}$. By contradiction, let l be a literal s.t. $l \notin \sigma_1$, $\bar{l} \notin \sigma_1$, and $l \in \sigma_0$ (since σ_0 is complete by hypothesis, if $l \notin \sigma_0$, we can still select $\bar{l}$). Then, the reduct P^A contains a rule

$$h(l, 1) \leftarrow h(l, 0). \tag{28}$$

Since A is closed under P^A, $h(l, 1) \in A$ and $l \in \sigma_1$. Contradiction.

Impossibility conditions respected. By contradiction, assume that condition $impossible\_if(a, [l_1, \ldots, l_m])$ is not respected. Then, $h(\{l_1, \ldots, l_m\}, 0) \subseteq A$ and $o(a, 0) \in A$. Therefore, the body of the α_d-mapping (15) of the impossibility condition is satisfied by A, and A is not a (consistent) answer set.

(27) holds. Let us prove that $\sigma_1 \supseteq E(a_0, \sigma_0)$. Consider a dynamic law d in SD of the form $causes(a, l_0, [l_1, \ldots, l_m])$, such that $\{l_1, \ldots, l_m\} \subseteq \sigma_0$ and $a \in a_0$. Since A is closed under $\alpha_d(d)$ (15), $h(l_0, 1) \in A$. Then, $\sigma_1 \supseteq E(a_0, \sigma_0)$.

$\sigma_1 \supseteq \sigma_1 \cap \sigma_0$ is trivially true.

Let us prove that σ_1 is closed under the static laws of SD. Consider a static law s, of the form $caused(l_0, [l_1, \ldots, l_m])$, such that $\{l_1, \ldots, l_m\} \subseteq \sigma_0$. Since A is closed under $\alpha_d(s)$ (15), $h(l_0, 1) \in A$.

Let us prove that σ_1 is the minimal set satisfying all conditions. By contradiction, assume that there exists a set $\delta \subset \sigma_1$ such that $\delta \supseteq E(a_0, \sigma_0) \cup (\sigma_1 \cap \sigma_0)$ and that δ is closed under the static laws of SD. We will prove that this implies that A is not an answer set of P.

Let A' be the set obtained by removing from A all atoms $h(l, 1)$ such that $l \in \sigma_1 \setminus \delta$. Since $\delta \subset \sigma_1$, $A' \subset A$.

Since $\delta \supseteq E(a_0, \sigma_0) \cup (\sigma_1 \cap \sigma_0)$, for every $l \in \sigma_1 \setminus \delta$ it must be true that $l \notin \sigma_0$ and $l \notin E(a_0, \sigma_0)$. Therefore there must exist (at least) one static law $caused(l, [l_1, \ldots, l_m])$ such that $\{l_1, \ldots, l_m\} \subseteq \sigma_1$ and $\{l_1, \ldots, l_m\} \not\subseteq \delta$. Hence, A' is closed under the rules of P^A. This proves that A is not an answer set of P. Contradiction. $\square$

We are now ready to prove the main result of this subsection. The following notation will be used in the theorems that follow. Let SD be an action description, and $M = \langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$ be a sequence where σ_i are sets of fluent literals and a_i are actions. $o(M)$ denotes

$$\bigcup_t o(a_t, t),$$

with $o(a, t)$ from (12). The length of M, denoted by $l(M)$, is $\frac{m-1}{2}$, where m is the number of elements of M.

Given an action description SD and a sequence $M = \langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$, $\beta_d^n(SD, M)$ denotes the program

$$\beta_d^n \cup h(\sigma_0, 0) \cup o(M). \tag{29}$$

We will use $\beta_d^n(M)$ as short form for $\beta_d^n(SD, M)$.

Lemma 5

Let SD be an action description, $M = \langle \sigma_0, a_0, \sigma_1, \ldots, a_{n-1}, \sigma_n \rangle$ be a sequence of length n, and $\beta_d^n(M)$ be defined as in (29). If σ_0 is a state, then, M is a trajectory of $\mathcal{T}(SD)$ iff M is defined by an answer set of $P = \beta_d^n(M)$.

12.1.2 Step 2

In this subsection, we extend the previous encoding in order to be able to generate programs whose answer sets describe exactly the paths consistent with a specified recorded history Γ_n.

Let $\Sigma_{\Gamma,d}^n(SD)$ denote the signature defined as follows:

- $const(\Sigma_{\Gamma,d}^n(SD)) = const(\Sigma_d^n(SD))$;
- $pred(\Sigma_{\Gamma,d}^n(SD)) = pred(\Sigma_d^n(SD)) \cup \{hpd, obs\}$.

Let

$$\alpha_d^n(SD, \Gamma_n) = \langle \Pi_d^\Gamma, \Sigma_{\Gamma,d}^n \rangle, \tag{30}$$

where

$$\Pi_d^\Gamma = \Pi_d^\beta(SD) \cup \hat{\Pi} \cup \Gamma_n. \tag{31}$$

$\Pi_d^\beta(SD)$ is defined as in (17), and $\hat{\Pi}$ is the set of rules:

$$\begin{array}{llll}
3. & o(A,T) & \leftarrow & hpd(A,T). \\
4. & h(L,0) & \leftarrow & obs(L,0). \\
5. & & \leftarrow & obs(L,T), \\
 & & & \text{not } h(L,T).
\end{array}$$

(Notice that these rules are equal the last 3 rules of program Π, defined in Section 4.)

When we refer to a single system description, we will often drop argument SD from $\Sigma_{\Gamma,d}^n(SD), \alpha_d^n(SD, \Gamma_n), \Pi_d^\Gamma(SD)$ in order to simplify the presentation.

Notice that, as we did before, in the rest of this section we will restrict attention to ground programs.

Proposition 1

If the initial situation of Γ_n is complete, i.e. for any fluent f of SD, Γ_n contains $obs(f, 0)$ or $obs(\neg f, 0)$, then

$$M \text{ is a model of } \Gamma_n \tag{32}$$

iff

$$M \text{ is defined by some answer set of } \alpha_d^n(SD, \Gamma_n). \tag{33}$$

12.1.3 Step 3

In this section we prove that models of Γ_n are in a one-to-one correspondence with the answer sets of $\alpha(SD, \Gamma_n)$. We will do this by proving that the answer sets of $\alpha_d^n(SD, \Gamma_n)$ and of $\alpha(SD, \Gamma_n)$ define the same models.

In order to prove this equivalence, we define a new encoding of $\mathcal{AL}$ that will allow us to link $\alpha(SD, \Gamma_n)$ and $\alpha_d^n(SD, \Gamma_n)$.

Let SD be an action description of $\mathcal{AL}$ and $\Sigma(SD)$ be its signature. For any positive integer n, $\Sigma^n(SD)$ denotes the signature obtained as follows:

- $const(\Sigma^n(SD)) = const(\Sigma(SD)) \cup \{0, \dots, n\} \cup \{1, \dots, k\}$, where k is the maximum number of preconditions present in the laws of SD;
- $pred(\Sigma^n(SD)) = \{h, o, d\_law, s\_law, head, action, prec, all\_h, prec\_h\}$.

Let

$$\alpha^n(SD) = \langle \Pi^\alpha(SD), \Sigma^n(SD) \rangle, \tag{34}$$

where

$$\Pi^\alpha(SD) = \bigcup_{r \in SD} \alpha(r) \tag{35}$$

and $\alpha(r)$ is defined as in Section 4.

Finally, let

$$\beta^n(SD) = \langle \Pi^\beta(SD), \Sigma^n(SD) \rangle, \tag{36}$$

where

$$\Pi^\beta(SD) = \Pi^\alpha(SD) \cup \Pi^- \tag{37}$$

and Π^- is the set of rules:

$$
\begin{array}{llll}
1. & h(L, T') & \leftarrow & d\_law(D), \\
 & & & head(D, L), \\
 & & & action(D, A), \\
 & & & o(A, T), \\
 & & & prec\_h(D, T). \\
2. & h(L, T) & \leftarrow & s\_law(D), \\
 & & & head(D, L), \\
 & & & prec\_h(D, T). \\
3. & all\_h(D, N, T) & \leftarrow & prec(D, N, nil). \\
4. & all\_h(D, N, T) & \leftarrow & prec(D, N, P), \\
 & & & h(P, T), \\
 & & & all\_h(D, N', T). \\
5. & prec\_h(D, T) & \leftarrow & all\_h(D, 1, T). \\
6. & h(L, T') & \leftarrow & h(L, T), \\
 & & & not\ h(\overline{L}, T'). \\
7. & & \leftarrow & h(L, T), h(\overline{L}, T).
\end{array}
$$

(Notice that these rules correspond to rules $(1)-(7)$ of program Π defined in Section 4.)

When we refer to a single action description, we will often drop the argument from $\Sigma^n(SD), \alpha^n(SD), \Pi^\alpha(SD), \beta^n(SD), \Pi^\beta(SD)$ in order to simplify the presentation. We will also restrict attention to the ground versions of the programs just defined. For this reason, we will abuse notation slightly and denote by $\alpha^n(SD)$ and $\beta^n(SD)$ the ground versions of the programs defined above.

The following theorem establishes a link between β^n and β^n_d.

Lemma 6
Let SD be an action description, n be a positive integer, and Q denote $lit(\beta^n) \setminus lit(\beta^n_d)$. Then, for any program R such that $lit(R) \cap Q = \emptyset$,

$$\beta^n \cup R \succ_Q \beta^n_d \cup R.$$

Proof
Let $\vec{q}$ be an ordering of the elements of Q, P be $\beta^n \cup R$, and P_d be $\beta^n_d \cup R$.

Notice that, in $e(P, \vec{q})$, the elements of Q only occur in the rules that define them, and that $lit(R) \cap Q = \emptyset$ by hypothesis. Then, by Lemma 3,

$$P \succ_Q t(P, \vec{q}). \tag{38}$$

It can also be easily checked that $t(P, \vec{q}) = P_d$. Hence, (38) can be rewritten as

$$P \succ_Q P_d,$$

that is,

$$\beta^n \cup R \succ_Q \beta^n_d \cup R.$$

□

We are finally able to give the proof of Theorem 1.

Theorem 1
If the initial situation of Γ_n is complete, i.e. for any fluent f of SD, Γ_n contains $obs(f, 0)$ or $obs(\neg f, 0)$, then M is a model of Γ_n iff M is defined by some answer set of $\alpha(SD, \Gamma_n)$.

Proof
By Proposition 1, M is a model of Γ_n iff M is defined by some answer set of $\alpha^n_d(SD, \Gamma_n)$. Let P_d be $\alpha^n_d(SD, \Gamma_n)$ and P be $\alpha(SD, \Gamma_n)$. Let R be $P \setminus \beta^n$. Let also Q be $lit(P) \setminus lit(P_d)$. By Lemma 6, $\beta^n \cup R \succ_Q \beta^n_d \cup R$. From this we obtain that $\alpha(SD, \Gamma_n) \succ_Q \alpha^n_d(SD, \Gamma_n)$. Notice that predicate names h and o are common to the signatures of both P and P_d. Then, the thesis follows from the definition of Strong Conservative Extension. □

The following corollary extends Theorem 1 to the case in which the initial situation of Γ_n is not complete.

Corollary 2
Let R be

$$
\begin{array}{rcl}
h(F,0) & \leftarrow & \text{not } h(\neg F,0). \\
h(\neg F,0) & \leftarrow & \text{not } h(F,0).
\end{array}
$$

For any history Γ_n,

$$M \text{ is a model of } \Gamma_n \tag{39}$$

iff

$$M \text{ is defined by some answer set of } \alpha(SD,\Gamma_n) \cup R. \tag{40}$$

Proof
Let σ_0 be the first component of M, and $obs(\sigma_0,0) = \{obs(l,0) \mid l \in \sigma_0\}$. First of all, we will show that (40) is equivalent to

$$M \text{ is defined by some answer set of } \alpha(SD,\Gamma_n) \cup obs(\sigma_0,0). \tag{41}$$

Let $\Pi_1 = \alpha(SD,\Gamma_n) \cup R$ and $\Pi_2 = \alpha(SD,\Gamma_n) \cup obs(\sigma_0,0)$. Consider a splitting of Π_1 and Π_2 based on set

$$S = \{obs(l,0) \mid l \in FL\} \cup \{h(l,0) \mid l \in FL\}$$

where FL is the set of fluent literals of SD. Let set Q include:

- the set of ground instances, with $T = 0$, of rules (2)-(5) (we need to consider only the instances where variable D denotes a static law), and (9) from program Π, in Section 4;
- the subset of $\alpha(SD)$ containing those facts occurring in the body of the above rules.

Π_1 and Π_2 are split by S so that:

$$bottom_S(\Pi_1) = R \cup Q \cup (\Gamma_n \cap S), \tag{42}$$
$$bottom_S(\Pi_2) = Q \cup obs(\sigma_0,0), \tag{43}$$
$$top_S(\Pi_1) = top_S(\Pi_2). \tag{44}$$

$bottom_S(\Pi_2)$ has a unique answer set, A_2, and $A_2 \cap lit(h) = h(\sigma_0,0)$. It can be shown that there exists an answer set, A_1, of $bottom_S(\Pi_1)$, such that $A_1 \subseteq A_2$. Moreover, for such A_1,

$$A_1 \setminus lit(obs) = A_2 \setminus lit(obs). \tag{45}$$

Let Q' denote the set of ground instances, with $T = 0$, of rule (10) from program Π in Section 4. From (44) and (45),

$$e_S(\Pi_1 \setminus Q', A_1) \simeq e_S(\Pi_2 \setminus Q', A_2). \tag{46}$$

Since the body of the rules in Q' is never satisfied,

$$e_S(\Pi_1, A_1) \simeq e_S(\Pi_2, A_2). \tag{47}$$

Let B be an answer set of $e_S(\Pi_2, A_2)$ and $C_2 = B \cup A_2$. By the Splitting Set

Theorem, $C_1 = B \cup A_1$ is an answer set of Π_1. This implies that (40) is equivalent to (41).

Now we will complete the proof by showing that (41) is equivalent to (39). Since

$$\alpha(SD, \Gamma_n) \cup obs(\sigma_0, 0) = \alpha(SD, \Gamma_n \cup obs(\sigma_0, 0)),$$

Equation (41) holds iff

$$M \text{ is defined by some answer set of } \alpha(SD, \Gamma_n \cup obs(\sigma_0, 0)). \tag{48}$$

By Theorem 1, (48) holds iff

$$M \text{ is a model of } \Gamma_n \cup obs(\sigma_0, 0). \tag{49}$$

By Definition 2.2(a), (49) holds iff

$$M \text{ is a model of } \Gamma_n. \tag{50}$$

□

12.2 Answer sets of $D_0(\mathcal{S})$ and candidate diagnoses

Theorem 2 establishes a link between answer sets and candidate diagnoses. In this section, we give a proof of this theorem.

Theorem 2

Let $\langle \Sigma, T, W \rangle$ be a diagnostic domain, SD be a system description of T, $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$ be a symptom of the system's malfunctioning, and E and δ be sets of ground atoms.

$$\langle E, \Delta \rangle \text{ is a candidate diagnosis of } \mathcal{S} \tag{51}$$

iff

$$\langle E, \Delta \rangle \text{ is determined by an answer set of } D_0(\mathcal{S}). \tag{52}$$

Proof

By definition of candidate diagnosis, (51) holds iff

$$\begin{array}{c} \text{there exists a model, } M, \text{ of history } \Gamma_n \cup O_n^m \cup E \text{ such that} \\ \Delta = \{c \mid M \models h(ab(c), m)\}. \end{array} \tag{53}$$

By Corollary 2, (53) holds iff

$$\begin{array}{c} \text{there exists an answer set, } X, \text{ of } P = Conf(\mathcal{S}) \cup E \text{ such that} \\ \Delta = \{c \mid h(ab(c), m) \in X\}. \end{array} \tag{54}$$

Consider now (52). By definition 4.2, (52) holds iff

$$\begin{array}{c} \text{there exists an answer set, } X', \text{ of } P' = D_0(\mathcal{S}) \text{ such that} \\ \langle E, \Delta \rangle \text{ is determined by } X'. \end{array} \tag{55}$$

Let

$$SP_0 = \{hpd(a, t) \mid hpd(a, t) \in P \text{ and } a \in A_e\},$$

$$SP = SP_0 \cup \{o(a,t) \mid hpd(a,t) \in SP_0\}.$$

SP is a splitting set for both P and P'.

By the Splitting Set Theorem, (54) and (55) are, respectively, equivalent to

$$\begin{array}{l}\text{(a) there exists an answer set, } X_B \text{, of } bottom_{SP}(P) \text{ and} \\ \quad \text{(b) there exists an answer set, } Y \text{, of } e_{SP}(P, X_B) \\ \quad\quad \text{such that } \Delta = \{c \mid h(ab(c), m) \in X_B \cup Y\}.\end{array} \tag{56}$$

$$\begin{array}{l}\text{(a) there exists an answer set, } X'_B \text{, of } bottom_{SP}(P') \text{ and} \\ \quad \text{(b) there exists an answer set, } Y' \text{, of } e_{SP}(P', X'_B) \\ \quad\quad \text{such that } \langle E, \Delta\rangle \text{ is determined by } X'_B \cup Y' \cdot\end{array} \tag{57}$$

We want to prove that (56) and (57) are equivalent.

Let

$$E_d = \{hpd(a,t) \mid hpd(a,t) \in E \text{ and } hpd(a,t) \notin \Gamma_n\},$$

and Q be the set of rules

$$o(A,T) \leftarrow hpd(A,T).$$

for A and T such that $hpd(A,T) \in E_d$. $bottom_{SP}(P)$ is $E_d \cup Q$ and $bottom_{SP}(P')$ is $DM_0 \cup Q$. Let B and B' be defined as follows:

$$\begin{array}{rl} H = & \{hpd(a,t) \mid hpd(a,t) \in E_d\} \\ B = & H \cup \{o(a,t) \mid hpd(a,t) \in H\} \\ B' = & B \setminus E_d \cup \{\neg o(a,t) \mid o(a,t) \notin B\}. \end{array} \tag{58}$$

Let us show that (56a) holds iff (57a) holds.

Assume that (56a) holds. It is easy to see that B is the unique answer set of $bottom_{SP}(P)$. If we observe that the answer sets of $bottom_{SP}(P')$ enumerate all possible sequences of exogenous actions, we obtain that B' is an answer set of $bottom_{SP}(P')$. Therefore, (57a) holds.

Now, assume that (57a) holds. As before, B' is an answer set of $bottom_{SP}(P')$. Immediately, we obtain that B is an answer set of $bottom_{SP}(P)$. Therefore, (56a) holds.

Let us now show that $e_{SP}(P,B) = e_{SP}(P',B')$. Notice that $top_{SP}(P) = Conf(\mathcal{S}) \setminus Q = top_{SP}(P')$. Let $I = B \cap B'$ and $\overline{I} = (B \cup B') \setminus I$. Observe that, for every literal $l \in \overline{I}$, $l \notin top_{SP}(P)$. This means that

$$e_{SP}(P,B) = e_{SP}(P,I) = e_{SP}(P',I) = e_{SP}(P',B'). \tag{59}$$

Let Z denote an answer set of $e_{SP}(P,B)$. By construction of B and B' and from (59),

$$\Delta = \{c \mid h(ab(c), n-1) \in (B \cup Z)\} \text{ iff } \langle E, \Delta\rangle \text{ is determined by } B' \cup Z \cdot$$

Hence, (56) and (57) are equivalent. $\square$

12.3 Properties of Diagnostic Module D_1

In this subsection, we show that using diagnostic module D_1 in place of D_0 is *safe*, i.e. D_1 will not miss any useful predictions about the malfunctioning components.

We start by introducing some terminology which is needed in the rest of the subsection.

Definition 12.2
Elementary action a_e is *relevant to fluent literal l* (written $rel(a_e, l)$), if:

- "$causes(a_e, l, P)$" $\in SD$;
- "$caused(l, [l_1, \ldots, l_m])$" $\in SD$ and a_e is relevant to some l_i from the preconditions of the law;
- "$impossible\_if(a'_e, [l_1, \ldots, l_m])$" $\in SD$, a'_e is relevant to some l, and a_e is relevant to some $\overline{l_i}$ from the preconditions of the condition.

An action, a, is *relevant to set O* of fluent literals if every elementary action from a is relevant to some $l \in O$.

Definition 12.3
The set, $rel(O)$, of fluent literals *relevant to collection of fluent literals O* is defined as follows.

1. $O \subseteq rel(O)$;
2. if "$causes(a_e, l, P)$" $\in SD$, and $l \in rel(O)$, then $P \subseteq rel(0)$;
3. if "$caused(l, P)$" $\in SD$, and $l \in rel(O)$, then $P \subseteq rel(0)$;
4. for every condition "$impossible\_if(a_e, [l_1, \ldots, l_m])$" from SD, if a_e is relevant to O, then $\{\overline{l_1}, \ldots, \overline{l_m}\} \subseteq rel(O)$.

Definition 12.4
States s_1,s_2 are called *equivalent w.r.t. a set of fluent literals O* ($s_1 \simeq_O s_2$) if

$$\forall l \in rel(O)\ l \in s_1 \text{ iff } l \in s_2.$$

Definition 12.5
The *rank of a sequence of actions, α, w.r.t. a collection of fluent literals, O*, is the number of elementary actions in α which are not relevant to O, and is denoted by $|\alpha|_O$.

Definition 12.6
Sequence of actions α_2 is the *reduct of* α_1 *w.r.t. O* ($\alpha_2 = red_O(\alpha_1)$) if α_2 is obtained from α_1 by replacing a by $a \setminus \{a_e\}$ if a_e is an exogenous action from a not relevant to O. We say that α_1 is *equivalent to* α_2 *w.r.t. O* ($\alpha_1 \simeq_O \alpha_2$) if $\alpha_1 = red_O(\alpha_2)$ or $\alpha_2 = red_O(\alpha_1)$.

Definition 12.7 (*Well-defined system description*)
Let T be the transition diagram corresponding to system description SD. SD is *well-defined* if, for any state s and action a, "$impossible\_if(a, P)$" $\in SD$ and $P \subseteq s$ iff there is no s' such that $\langle s, a, s' \rangle \in T$.

Definition 12.8 (*Set of final states*)
Let O be a collection of fluent literals. The set of final states w.r.t. O is

$$F_O = \{s \mid O \subseteq s\}.$$

Definition 12.9
A *relevant* candidate diagnosis of a symptom, $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$, is a candidate diagnosis, $\langle E, \Delta \rangle$, of $\mathcal{S}$ such that all actions in E are relevant to the observations in O_n^m.

In the theorems that follow, we will implicitly consider only well-defined system descriptions. We will also write $\langle s_0, \alpha, s_1 \rangle \in T$, to indicate that transition diagram T contains a path from state s_0 to state s_1 whose actions are labeled by sequence of compound actions α.

Lemma 7
Let O be a collection of fluent literals, s_0, s_0' be states such that $s_0 \simeq_O s_0'$, and α be a sequence of actions s.t. every action of α is relevant to O. If $\langle s_0, \alpha, s_1 \rangle \in T$, then there exists s_1' such that

1. $\langle s_0', \alpha, s_1' \rangle \in T$;
2. $s_1' \simeq_O s_1$.

Lemma 8
Let O be a collection of fluent literals, and $e, a_1, \dots, a_k$ be elementary actions. If e is not relevant to O and $\langle s_0, \{e, a_1, \dots, a_k\}, s_1 \rangle \in T$, then there exists s_1' such that

1. $\langle s_0, \{a_1, \dots, a_k\}, s_1' \rangle \in T$;
2. $s_1 \simeq_O s_1'$.

Theorem 4
Let SD be a system description and O be a collection of fluent literals. For every path $\langle s_0, \alpha, s_f \rangle$ from $\mathcal{T}(SD)$ and $s_0' \simeq_O s_0$ there is a path $\langle s_0', \alpha', s_f' \rangle$ such that:

$$\alpha \simeq_O \alpha',\ |\alpha'|_O = 0; \tag{60}$$

$$s_f' \simeq_O s_f. \tag{61}$$

Proof
By induction on the rank of α w.r.t. O, $|\alpha|_O$. In the rest of this proof, we will use T to denote $\mathcal{T}(SD)$.

Base case: $|\alpha|_O = 0$. By Lemma 7, there exists a path $\langle s_0', \alpha, s_f' \rangle \in T$ such that $s_f' \simeq_O s_f$.

Inductive step: since $|\alpha|_O > 0$, at least one elementary action irrelevant to O occurs in α. Hence, there exist states s_k, s_{k+1}, elementary actions $e_1, \dots, e_m$, and (possibly empty) sequences of actions α_1, α_2 such that:

1. all actions in α_1 are relevant to O, and $\langle s_0, \alpha_1, s_k \rangle \in T$;
2. e_1 is not relevant to O;

3. $\langle s_k, \{e_1, \dots, e_m\}, s_{k+1}\rangle \in T$;
4. $\langle s_{k+1}, \alpha_2, s_f\rangle \in T$.

First we show that there exists s'_{k+1} such that

$$\langle s'_0, \alpha_1\{e_2, \dots, e_m\}, s'_{k+1}\rangle \in T. \tag{62}$$

and $s'_{k+1} \simeq_O s_{k+1}$.

Let s'_0 be a state such that $s'_0 \simeq_O s_0$. From condition (1) and the inductive hypothesis, it follows that there exists $s'_k \simeq_O s_k$ such that

$$\langle s'_0, \alpha_1, s'_k\rangle \in T. \tag{63}$$

Next, we notice that, by conditions (2), (3) and Lemma 8, there exists s''_{k+1} such that $s''_{k+1} \simeq_O s_{k+1}$ and $\langle s_k, \{e_2, \dots, e_m\}, s''_{k+1}\rangle \in T$. By inductive hypothesis, there exists s'_{k+1} such that $s'_{k+1} \simeq_O s''_{k+1}$ and

$$\langle s'_k, \{e_2, \dots, e_m\}, s'_{k+1}\rangle \in T. \tag{64}$$

Since relation "$\simeq_O$" is transitive,

$$s_{k+1} \simeq_O s'_{k+1}. \tag{65}$$

Hence, (62) is proven.

Notice that, by construction, $|\alpha_2|_O < |\alpha|_O$. By condition (4), equation (65), and the inductive hypothesis, we obtain that there exist s''_f and α'_2 such that $s'_f \simeq_O s_f$, $\alpha_2 \simeq_O \alpha'_2$, $|\alpha'_2|_O = 0$, and

$$\langle s'_{k+1}, \alpha'_2, s''_f\rangle \in T. \tag{66}$$

Let γ be the sequence consisting of α_1, $\{e_2, \dots, e_m\}$ and α'_2. From (63)–(66), it follows that

$$\langle s'_o, \gamma, s''_f\rangle \in T. \tag{67}$$

By the inductive hypothesis there exists a path $\langle s'_0, \alpha', s'_f\rangle \in T$ such that $\alpha' \simeq_O \gamma$ and $s'_f \simeq_O s''_f$. (60) and (61) follow immediately from the transitivity of relation "$\simeq_O$". $\square$

The following theorem shows that, if only relevant candidate diagnoses are computed, no useful prediction about the system's malfunctioning is missed.

Theorem 5

Let $\langle \Sigma, T, W\rangle$ be a diagnostic domain, SD be a system description of T, and $\mathcal{S} = \langle \Gamma_n, O\rangle$ be a symptom of the system's malfunctioning. For every candidate diagnosis $D = \langle E, \Delta\rangle$ of $\mathcal{S}$ there exists a relevant candidate diagnosis $D_r = \langle E_r, \Delta_r\rangle$ such that $E_r = red_O(E)$.

Proof
First, let us prove that D_r is a candidate diagnosis. By definition of candidate diagnosis (Definition 3.1), E describes one or more paths in the transition diagram, whose final state, s_f, is consistent with O. By Theorem 4, E_r describes paths whose final state, s'_f, is equivalent to s_f w.r.t. O. Hence, s'_f is consistent with O, as well, and therefore D_r is a candidate diagnosis.

The fact that D_r is a relevant candidate diagnosis follows directly from Definition 12.9. $\square$

The next theorem proves that diagnostic module $D_1(\mathcal{S})$ generates all relevant candidate diagnoses of $\mathcal{S}$.

Theorem 6
Let $\langle \Sigma, T, W \rangle$ be a diagnostic domain, and *SD* be a system description of T. For every symptom of the system's malfunctioning, $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$, diagnostic module $D_1(\mathcal{S})$ computes all relevant candidate diagnoses of $\mathcal{S}$.

Proof
(Sketch.) By Theorem 2, $D_0(\mathcal{S})$ computes all candidate diagnoses of $\mathcal{S}$. $D_1(\mathcal{S})$ essentially consists in the addition of a constraint. This constraint makes the module reject all candidate diagnoses which are not relevant to the observations in O_n^m. Hence, all candidate diagnoses returned by $D_1(\mathcal{S})$ are relevant to $\mathcal{S}$. $\square$

12.4 Properties of *Find_Diag*

The properties of *Find_Diag* are described by Theorem 3. In order to prove the theorem, we prove separately the termination of the algorithm and its correctness.

Lemma 9
Let $\langle \Sigma, T, W \rangle$ be a diagnostic domain, *SD* be a system description of T, and $\mathcal{S} = \langle \Gamma_n, O_n \rangle$ be a symptom of the system's malfunctioning. Then, *Find_Diag*$(\mathcal{S})$ terminates.

Lemma 10
Let *SD* be defined as above, $\mathcal{S}_0 = \langle \Gamma_n, O_n^m \rangle$ be a symptom of the system's malfunctioning, and $\langle E, \Delta \rangle =$ *Find_Diag*$(\mathcal{S})$, where the value of variable $\mathcal{S}$ is set to $\mathcal{S}_0$. If $\Delta \neq \emptyset$, then

$\langle E, \Delta \rangle$ is a diagnosis of $\mathcal{S}_0$;

otherwise, $\mathcal{S}_0$ has no diagnosis.

Proof
Let us show that, if $\Delta = \emptyset$, $\mathcal{S}_0$ has no diagnosis. By definition of candidate diagnosis and Theorem 2, *Candidate_Diag* returns $\Delta = \emptyset$ only if $\mathcal{S}_0$ has no diagnosis. The proof is completed by observing that, if *Candidate_Diag* returns $\Delta = \emptyset$, the function terminates immediately, and returns $\langle E, \Delta \rangle$.

Let us now assume that $\Delta \neq \emptyset$. We have to show that

1. $\langle E, \Delta \rangle$ is a candidate diagnosis of $\mathcal{S}_0$, and
2. all the components in Δ are faulty.

Let O^i $(i \geq 1)$ denote the value of variable O at the beginning of the i^{th} iteration of the *repeat* ... *until* loop of *Find_Diag*. Notice that O^1 is equal to the initial value of O_n. By Lemma 9, the algorithm terminates. Let u denote the index of the last iteration of the *repeat* ... *until* loop.

Since $\Delta \neq \emptyset$, by Theorem 2 and by the fact that $\langle E, \Delta \rangle$ is not updated by the while loop,

$$\langle E, \Delta \rangle \text{ is a candidate diagnosis of } \langle \Gamma_n, O^u \rangle.$$

We want to show that this implies statement 1.

By the definition of candidate diagnosis,

$$\begin{array}{c} \text{there exists a model, } M\text{, of } \Gamma_n \cup O^u \cup E \text{ such that} \\ \Delta = \{c \mid M \models h(ab(c), m)\}. \end{array} \tag{68}$$

Let M denote one such model. From Corollary 2, (68) holds iff

$$\begin{array}{c} \text{there exists an answer set, } \mathcal{AS}\text{, of} \\ P = \alpha(SD, \Gamma_n \cup O^u \cup E) \cup R \text{ such that} \\ \Delta = \{c \mid h(ab(c), m) \in \mathcal{AS}\}. \end{array} \tag{69}$$

Let A^u denote one such answer set.

Since $\mathcal{S}_0$ is a symptom, $n > 0$. Notice that $O' = O^u \setminus O^1$ is a set of observations made at time n. Let C denote the set of constraints of P of the form

$$\begin{array}{rl} \leftarrow & obs(l, t), \\ & \text{not } h(l, t). \end{array} \tag{70}$$

where $obs(l, t) \in O'$ – these constraints correspond to rule (10) of Π (see Section 4)). Let also Q denote $P \setminus C$.

By the properties of the answer set semantics, (69) holds iff

$$\begin{array}{c} A^u \text{ is an answer set of } Q, A^u \text{ does not violate } C, \text{ and} \\ \Delta = \{c \mid h(ab(c), m) \in A^u\}. \end{array} \tag{71}$$

Notice that O' is a splitting set for Q, and $bottom_{O'}(Q) = O'$. Since no literal of O' occurs in $top_{O'}(Q)$, $e_{O'}(Q, O') = Q \setminus O'$. Let R denote $Q \setminus O'$. By the Splitting Set Theorem, A^u is an answer set of Q iff $A^u \setminus O'$ is an answer set of R.

Let A^1 denote $A^u \setminus O'$. Observe that the literals of O' occur, within P, only in the constraints of C, and that they never occur under negation as failure. Therefore, if A^u does not violate C, then A^1 does not violate C, either. Hence, (71) implies that

$$\begin{array}{c} A^1 \text{ is an answer set of } R, A^1 \text{ does not violate } C, \text{ and} \\ \Delta = \{c \mid h(ab(c), m) \in A^1\}. \end{array} \tag{72}$$

By the properties of the answer set semantics, (72) holds iff

$$\begin{array}{c} A^1 \text{ is an answer set of } R \cup C, \text{ and} \\ \Delta = \{c \mid h(ab(c), m) \in A^1\}. \end{array} \tag{73}$$

Since $R = Q \setminus O'$,

$$R \cup C = Q \setminus O' \cup C = P \setminus O' = \alpha(SD, \Gamma_n \cup O^1 \cup E) \cup R.$$

Hence (73) can be rewritten as:

$$\begin{array}{c} A^1 \text{ is an answer set of } \alpha(SD, \Gamma_n \cup O^1 \cup E) \cup R, \text{ and} \\ \Delta = \{c \mid h(ab(c), m) \in A^1\}. \end{array} \tag{74}$$

From Corollary 2, (74) holds iff

$$\begin{array}{c} \text{there exists a model, } M, \text{ of } \Gamma_n \cup O^1 \cup E \text{ such that} \\ \Delta = \{c \mid M \models h(ab(c), n-1)\}. \end{array} \tag{75}$$

By the definition of candidate diagnosis,

$\langle E, \Delta \rangle$ is a candidate diagnosis of $\mathcal{S}_0$.

To prove statement 2, notice that Δ is the result of the latest call to *Candidate_Diag*. Since, by hypothesis, $\Delta \neq \emptyset$, the value of variable *diag* at the end of the final iteration of the *repeat* ... *until* loop must have been *true*. In turn, this implies that, in the same iteration, the while loop terminated with $\Delta_0 = \emptyset$ and $diag = true$. Therefore all the components in Δ are faulty and, by definition of diagnosis,

$\langle E, \Delta \rangle$ is a diagnosis of $\mathcal{S}_0$.

□

Theorem 3
Let $\langle \Sigma, T, W \rangle$ be a diagnostic domain, *SD* be a system description of T, and $\mathcal{S} = \langle \Gamma_n, O_n^m \rangle$ be a symptom of the system's malfunctioning. Then,

1. *Find_Diag*($\mathcal{S}$) terminates;
2. Let $\langle E, \Delta \rangle = Find\_Diag(\mathcal{S})$, where the value of variable $\mathcal{S}$ is set to $\mathcal{S}_0$. If $\Delta \neq \emptyset$, then

   $\langle E, \Delta \rangle$ is a diagnosis of $\mathcal{S}_0$;

   otherwise, $\mathcal{S}_0$ has no diagnosis.

Proof
Statement 1 is proven by applying Lemma 9. Statement 2 is proven by applying Lemma 10. □

References

Baral, C., and Gelfond, M. Reasoning agents in dynamic domains. In Minker, J,. ed., *Logic-Based Artificial Intelligence*, Kluwer Academic Publishers, (2000), 257–279,

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

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.

Brass, S., and Dix, J. A Characterization of the Stable Semantics by Partial Evaluation. In *Proceedings of the 10th Workshop on Logic Programming*, Zuerich, Oct. 1994.

Buccafurri, F., Leone, N., and Rullo, P. Strong and Weak Constraints in Disjunctive Datalog In Dix, Furbach, and Nerode, eds., *Proceedings of the 4th International Conference on Logic Programming and Non-Monotonic Reasoning (LPNMR'97)*, number 1265 in Lecture Notes in AI (LNAI), 2–17, Dagstuhl, Germany, July 1997. Springer Verlag.

Citrigno, S., Eiter, T., Faber, W., Gottlob, G., Koch, C., Leone, N., Mateis, C., Pfeifer, G, and Scarcello, F. The dlv system: Model generator and application frontends. In *Proceedings of the 12th Workshop on Logic Programming*, 128–137, 1997.

Cholewinski, P., Marek, W., and Truszczynski, M. Default Reasoning System DeReS. In *International Conference on Principles of Knowledge Representation and Reasoning*, 518–528, Morgan Kauffman, 1996.

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.

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

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

Gelfond, M., and Son, T. C. Reasoning with Prioritized Defaults. In J.Dix, L. M. Pereira, T. Przymusinski eds, *Lecture Notes in Artificial Intelligence, 1471*, 164–224, 1998.

Kowalski, R., and Sadri, F. From logic programming towards multi-agent systems. In *Annals of Mathematics and Artificial Intelligence*, 25, 391–419, 1999.

Lifschitz, V. On the logic of causal explanation. *Artificial Intelligence*, 96:451–465, 1997.

Lifschitz, V. Action languages, Answer Sets, and Planning. In *The Logic Programming Paradigm: a 25-Year Perspective*. 357–373, Springer Verlag, 1999.

Lifschitz, V., and Turner., H. Splitting a logic program. In Pascal Van Hentenryck, editor, *Proc. of the Eleventh Int'l Conf. on Logic Programming*, 23–38, 1994.

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.

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

McCain, N., and Turner, H. Causal theories of action and change. In *Proc. of AAAI*, 460–465, 1997.

McCarthy, J., and Hayes, P. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 4, 463–502. Edinburgh University Press, Edinburgh, 1969.

McIlraith, T. Representing actions and state constraints in model-based diagnosis. In *Proc. of AAAI97*, 43–49, 1997.

McIlraith, T. Explanatory diagnosis conjecturing actions to explain observations. In *Proceedings of the 1998 KR Conference*, 167–177, 1998.

McIlraith, S., and Scherl, R. What Sensing Tells Us: Towards a Formal Theory of Testing for Dynamical Systems In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'2000)*, 483–490, August, 2000.

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.

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.

Otero, R., and Cabalar, P. Pertinence and Causality In *Working Notes of 3rd Workshop on Nonmonotonic Reasoning, Action, and Change (NRAC)*, 111–119, Stockholm 1999.

Otero, M., and Otero, R. Using causality for diagnosis In *Proc. of 11th Int. Workshop on Principles of Diagnosis*, DX-80, 171–176, Mexico, 2000.

Reiter, R. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*, 119–140. Plenum Press, New York, 1978.

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

Shanahan, M. *Solving the frame problem: A mathematical investigation of the commonsense law of inertia*. MIT press, 1997.

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

Thielscher, M. Ramification and causality. *Artificial Intelligence*, 89(1-2):317–364, 1997.

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

Thielscher, M. Introduction to Fluent Calculus In *Linkoping Electronic Articles in Computer and Information Sciences*, vol 3(14), http://www.ep.lin.se/ea/cis/1998/014.

Turner, H. Splitting a Default Theory, *In Proc. of AAAI-96*, 645–651, 1996.

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