

# **Answer Set Based Design of Autonomous, Rational Agents**

Marcello Balduccini

Knowledge Representation Lab  
Computer Science Department  
Texas Tech University

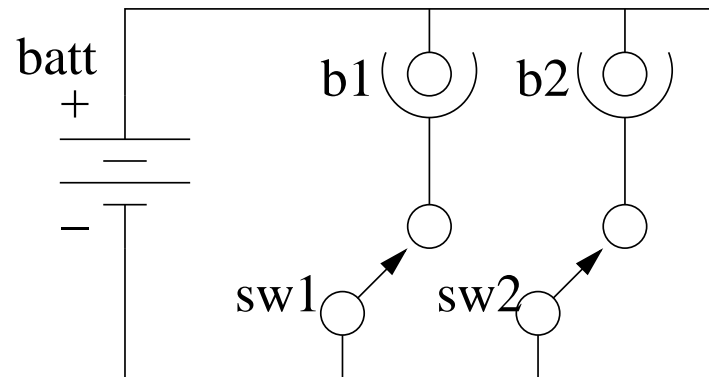
November 18, 2005

# Our Goal

To design an agent capable of  
rational, autonomous interaction with the environment.

## **Example of Agent Behavior**

# A Physical System



## Domain Properties

- $closed(SW)$
- $lit(Bulb)$
- $ab(Bulb)$
- $ab(batt)$

## Agent Actions

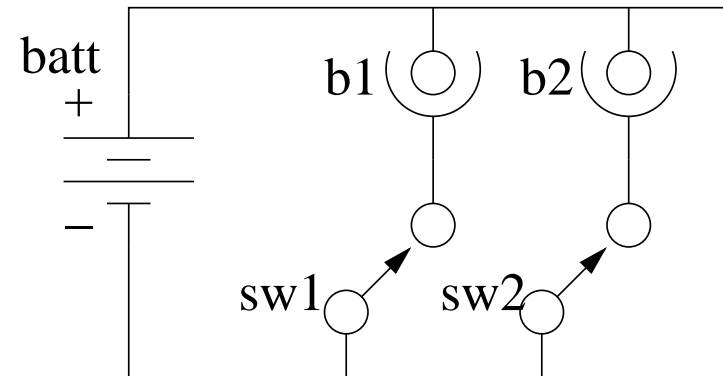
- $flip(SW)$
- $replace(Bulb)$
- $replace(batt)$

## Exogenous Actions

- $blow\_up(Bulb)$

# Planning

**Agent's goal:**  $lit(b_1)$

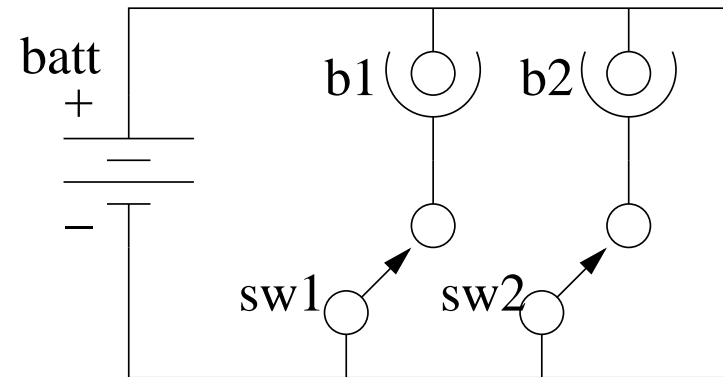


- **Observes:** switches open; bulbs off; components ok
- **Finds plan:**  $flip(sw_1)$
- **Executes:**  $flip(sw_1)$
- **Observes:** ...?

# Diagnosis

[...]

- **Executes:**  $flip(sw_1)$



- **Observes:**  $\neg lit(b_1)$   $\Leftarrow$  **UNEXPECTED!!!**
- **Explains:**  $blow\_up(b_1)$  occurred *concurrently* with  $flip(sw_1)$
- **Tests:** is  $ab(b_1)$  true?
- **Answer:** ...?

## Recovery

[...]

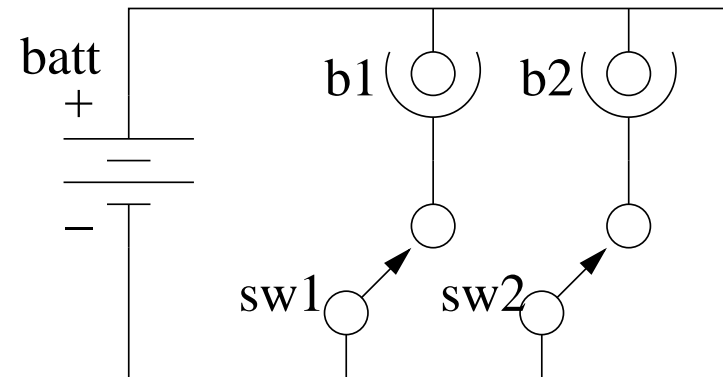
- **Tests:** is  $ab(b_1)$  true?

- **Answer:**  $ab(b_1)$  true

- **Finds plan:**  $replace(b_1)$

- **Executes:**  $replace(b_1)$

- **Observes:**  $lit(b_1)$   $\Leftarrow$  **SUCCESS!!!**

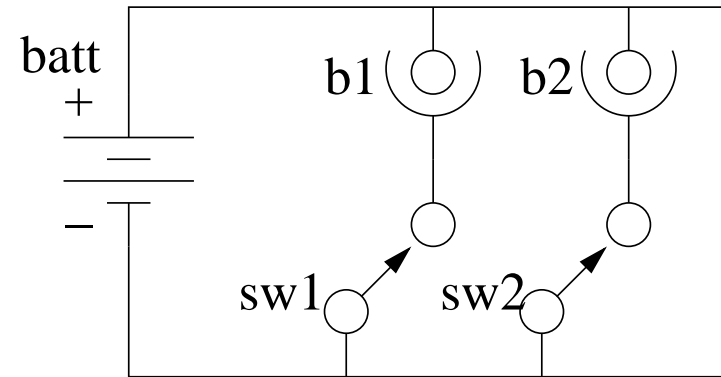


# **Beyond Diagnosis**



## What if...?

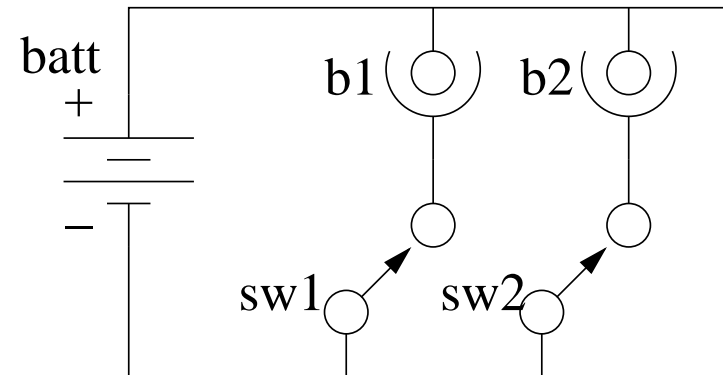
Receives new goal:  $lit(b_2)$



- Finds plan:  $flip(sw_2)$
- Executes:  $flip(sw_2)$
- Observes:  $\neg lit(b_2)$   $\Leftarrow$  **UNEXPECTED!!!**
- Explains:  $blow\_up(b_2)$  occurred (e.g. with  $flip(sw_2)$ )
- Tests: is  $ab(b_2)$  true?
- Answer:  $ab(b_2)$  false!!!  $\Leftarrow$  **NO DIAGNOSES LEFT**

# Learning

[...]



- **Explains:** “if  $sw_1$ ,  $sw_2$  are closed,  $batt$  becomes faulty”
- **Tests:** is  $ab(batt)$  true?
- **Answer:**  $ab(batt)$  true
- **Finds plan:** ...?

# Recovery

[...]

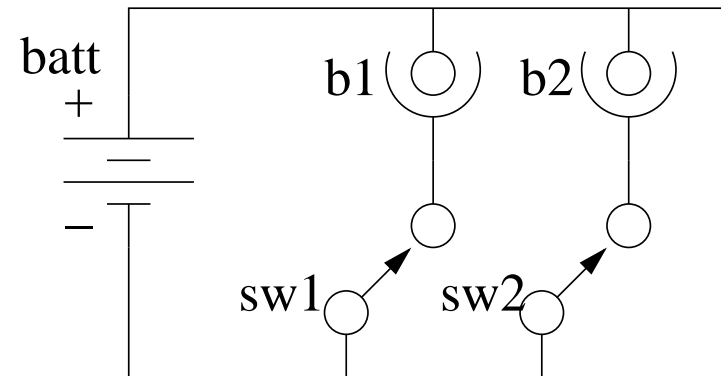
- **Finds plan:**  $flip(sw_1); replace(batt)$

- **Executes:**  $flip(sw_1)$

- **Observes:**  $sw_1$  open

- **Executes:**  $replace(batt)$

- **Observes:**  $lit(b_2)$   $\Leftarrow$  **SUCCESS!!!**



# How Do We Build It?

## Key Elements

- Control loop
  - ◇ Simple procedural code
- Domain model
  - ◇ Encoded in action language  $\mathcal{AL}$ ;  
automatically translated to A-Prolog
- Reasoning modules
  - ◇ Written in A-Prolog

⇒ Features:

- ◇ Reasoning modules, control loop provably correct
- ◇ Writing domain models, reasoning modules,  
control knowledge: easy

# Control Loop: Observe-Think-Act loop

1. observe the world;
2. interpret the observations (*if needed*):
  - ◇ diagnose;
  - ◇ learn;
3. select a goal;
4. plan;
5. execute part of the plan.

# Domain Model

## Action Description $AD$

%% Flipping  $SW$  causes  $SW$  to become  
%% closed if it was open and vice-versa.  
%%

$d_1 : flip(SW)$  causes  $closed(SW)$  if  $\neg closed(SW)$ .

$d_2 : flip(SW)$  causes  $\neg closed(SW)$  if  $closed(SW)$ .

$s_1 : lit(b_1)$  if  $closed(sw_1), \neg ab(b_1)$ .

[...]

$d_3 : blow\_up(B)$  causes  $ab(B)$ .

$d_4 : replace(batt)$  causes  $\neg ab(batt)$ .

[...]



## Recorded History $H^{cT}$

### Initial Situation:

$obs(\neg closed(sw_1), 0), obs(\neg closed(sw_2), 0),$   
 $obs(\neg lit(b_1), 0), obs(\neg lit(b_2), 0),$   
 $obs(\neg ab(b_1), 0), obs(\neg ab(b_2), 0),$   
 $obs(\neg ab(batt), 0),$

### Agent Actions at step 0:

$hpd(flip(sw_1), 0),$

### Observations at step 1:

$obs(closed(sw_1), 1), obs(\neg lit(b_1), 1)$

# A-Prolog

## Language Features

- Knowledge representation language
- Roots: logic programming, non-monotonic reasoning
- Intuitive reading of statements closely matches formal semantics
- ◇ High-level specification language, *but also...*
- ◇ ...close to implementation level
- Programs are compact and easy to understand.

## Simple Examples

“If it is raining and you do not have a rain coat, take an umbrella.”

$$take\_umbrella \leftarrow raining, \neg have\_raincoat.$$

“It is raining. You do not have a rain coat.”

$$raining. \quad \neg have\_raincoat.$$

**Conclusion:**  $take\_umbrella \Leftarrow$  the agent take an umbrella.

## Simple Examples

“If I am a good student, I do not have any B’s.”

$\neg have\_B \leftarrow good\_student.$

“I have B’s, but I am a good student.”

$have\_B. \quad good\_student.$

**Contradiction:** conclusion  $\neg have\_B$  contradicts  $have\_B$ .

The program is **inconsistent**.

## A-Prolog with Variables

“If switch  $SW$  is closed,  $SW$  is connected to bulb  $B$ , and  $B$  is not malfunctioning, then  $B$  is lit.”

$$lit(B) \leftarrow closed(SW), connected(SW, B), \neg ab(B).$$

“Switches  $sw_1$ ,  $sw_2$ ,  $sw_3$  are closed and connected to  $b_1$ ,  $b_2$ ,  $b_3$ , respectively. Only  $b_2$  is malfunctioning.

$$\begin{aligned} &closed(sw_1). \quad closed(sw_2). \quad closed(sw_3). \\ &connected(sw_1, b_1). \quad connected(sw_2, b_2). \quad connected(sw_3, b_3). \\ &\neg ab(b_1). \quad ab(b_2). \quad \neg ab(b_3). \end{aligned}$$

**Answer Set:**  $\{lit(b_1), lit(b_3)\}$ .

## Set Notation

“If you behave, some of these toys may be yours.”

$$\{X \mid \textit{have}(X)\} \subseteq \{X \mid \textit{toy}(X)\} \leftarrow \textit{behave}.$$

Given facts: *behave*, *toy*(*t*<sub>1</sub>), *toy*(*t*<sub>2</sub>)

**Answer Sets:**

$$\begin{array}{l} \{\textit{have}(t_1), \textit{have}(t_2)\} \\ \{\textit{have}(t_1)\} \\ \{\textit{have}(t_2)\} \\ \{\} \end{array} \Leftarrow \textit{agent does not get any toys}$$

Abbreviation:  $\{\textit{have}(X) : \textit{toy}(X)\} \leftarrow \textit{behave}.$

## Translation of $AD$ in A-Prolog, $\alpha(AD)$

### Dynamic Law of $\mathcal{AL}$

$d_1 : flip(sw_1)$  causes  $closed(sw_1)$  if  $\neg closed(sw_1)$ .

### $\alpha$ -Translation:

```
{ %  $d_1$  is a dynamic law  
   $d_{law}(d_1)$ .  
  
  % The head of  $d_1$  is  $closed(sw_1)$   
   $head(d_1, closed(sw_1))$ .  
  
  % The action of  $d_1$  is  $flip(sw_1)$   
   $action(d_1, flip(sw_1))$ .  
  
  % Precondition #1 of  $d_1$  is  $\neg closed(sw_1)$   
   $prec(d_1, 1, \neg closed(sw_1))$ .
```



# Translating State Constraints

**Law:**

$lit(b_1)$  if  $closed(sw_1), \neg ab(b_1)$

**$\alpha$ -Translation:**

{  
  %  $s_1$  is a state constraint  
   $slaw(s_1)$ .  
  
  % The head of  $s_1$  is  $lit(b_1)$   
   $head(s_1, lit(b_1))$ .  
  
  % The preconditions of  $s_1$  are  $closed(sw_1)$  and  $\neg ab(b_1)$   
   $prec(s_1, 1, closed(sw_1))$ .  
   $prec(s_1, 2, \neg ab(b_1))$ .  
}

# Projecting the Effects of Actions

$$\begin{aligned} \text{holds}(L, T + 1) \leftarrow & \quad \text{dlaw}(D), \\ & \text{head}(D, L), \\ & \text{action}(D, A), \\ & \text{occurs}(A, T), \\ & \text{all\_prec\_hold}(D, T). \end{aligned}$$

...

$$\begin{aligned} \text{prec\_n\_holds}(D, N, T) \leftarrow & \quad \text{prec}(D, N, P), \\ & \text{holds}(P, T). \end{aligned}$$

# Planning

# Overview

- Agent's Goal: set of fluent literals, e.g.

$\{ \textit{have}(\textit{lots\_of}(\textit{money})), \neg \textit{in}(\textit{jail}) \}.$

- Approach: generate and test.
- *Generation*: possible occurrences of actions are generated.
- *Testing*: constraint ensuring that solutions achieve the goal.

## Planning Module

Consists of  $\alpha(\langle AD, H^{cT} \rangle)$  together with:

$PGEN :$  {

- %% select occurrences of actions for each step  
 $\{occurs(A, T) : ag\_action(A)\} \leftarrow T \geq cT.$
- %% goal achieved if required literals eventually hold  
 $goal\_achieved \leftarrow holds(g_1, T),$   
 $\dots,$   
 $holds(g_m, T).$
- %% plans achieve the goal  
 $\leftarrow not\ goal\_achieved.$

## Example

- $H^{cT}$ :  $\left\{ \begin{array}{l} obs(\neg closed(sw_1), 0), \quad obs(\neg closed(sw_2), 0), \\ obs(\neg lit(b_1), 0), \quad obs(\neg lit(b_2), 0), \\ obs(\neg ab(b_1), 0), \quad obs(\neg ab(b_2), 0), \\ obs(\neg ab(batt), 0) \end{array} \right.$

- Goal:  $\{lit(b_1)\}$ .

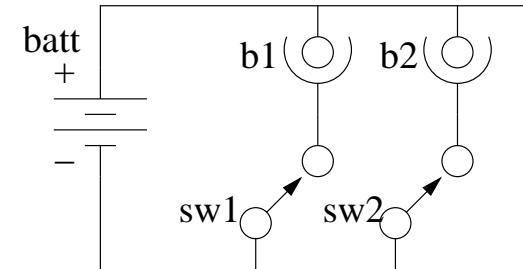
- Generation: possible sequence of actions is:

$occurs(flip(sw_1), 0).$

- Testing: according to the model,  $occurs(flip(sw_1), 0)$  yields the effect

$holds(lit(b_1), 1).$

**PLAN FOUND!!**



# Diagnosis

## Basics

- *Symptom*: history  $H^{cT}$  with unexpected observations
- $H^{cT}$  is symptom if:

$\alpha(\langle AD, H^{cT} \rangle)$  is inconsistent

- *Explanation*  $E$ : set of statements  $hpd(a_e, t)$  such that

$\alpha(\langle AD, H^{cT} \cup E \rangle)$  is consistent.

- *Candidate Diagnosis*:  $cD = \langle E, \Delta_E \rangle$ , where:
  - ◊  $E$ : explanation
  - ◊  $\Delta_E$ : components that may be damaged by actions of  $E$ .



# Diagnosis

- Approach: generate and test.
- *Generation*: possible occurrences of actions in the past are generated.
- *Testing*: sequences that do not explain the observations are discarded.

## Diagnostic Module

Consists of  $\alpha(\langle AD, H^{cT} \rangle)$  together with:

%% Select occurrences of actions for each step in the past  
 $\{ \text{occurs}(A, T) : \text{ex\_action}(A) \} \leftarrow 0 \leq T < cT.$

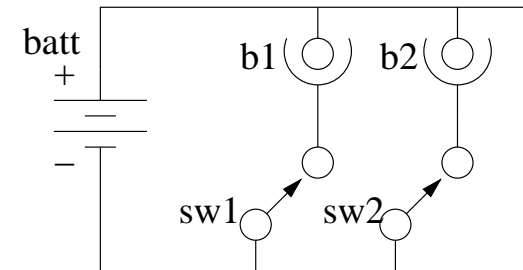
%% It is impossible for a prediction to  
%% disagree with an observation.

$\leftarrow \text{holds}(F, T), \text{obs}(\neg F, T).$

$\leftarrow \text{holds}(\neg F, T), \text{obs}(F, T).$

## Example: Diagnosing the Circuit

- $H^{cT}$ :  $\left\{ \begin{array}{l} obs(\neg closed(sw_1), 0), \quad obs(\neg closed(sw_2), 0), \\ obs(\neg lit(b_1), 0), \quad obs(\neg lit(b_2), 0), \\ obs(\neg ab(b_1), 0), \quad obs(\neg ab(b_2), 0), \quad obs(\neg ab(batt), 0) \\ \\ hpd(flip(sw_1), 0) \\ \\ obs(\neg lit(b_1), 1) \end{array} \right.$



- $\alpha(\langle AD, H^{cT} \rangle)$  inconsistent  $\Rightarrow H^{cT}$  is symptom
- Generation: possible sequence of actions is:

$occurs(blow\_up(b_1), 0)$

- Testing: according to the model,  $occurs(blow\_up(b_1), 0)$  justifies:

$obs(\neg lit(b_1), 1).$

**CANDIDATE DIAGNOSIS FOUND!!**

# Learning

## Modification Statements

- *Modification Statements*:  $d\text{law}(w)$ ,  $s\text{law}(w)$ ,  $\text{head}(w, l)$ ,  $\text{action}(w, a_e)$ ,  $\text{prec}(w, n, p)$ .
- *Valid set of Modification Statements*,  $Mod$ :
  - ◇ for every  $w$ ,  $s\text{law}(w)$  and  $d\text{law}(w)$  cannot be both in  $Mod$ ;
  - ◇ one  $\text{head}(w, l)$  statement for every  $s\text{law}(w)$  or  $d\text{law}(w)$  in  $Mod$ ;
  - ◇ one  $\text{action}(w, l)$  statement for every  $d\text{law}(w)$  in  $Mod$ ;

### Examples

$\{s\text{law}(w), \text{head}(w, l_1), \text{prec}(w, 1, l_2)\}$  is valid;

$\{s\text{law}(w), \text{prec}(w, 1, l_2)\}$  is not valid (missing *head*);

$\{d\text{law}(w), \text{head}(w, l_1), \text{prec}(w, 1, l_2)\}$  is not valid (missing *action*).

## Candidate Correction

- $upd(AD, Mod)$ : Update of  $AD$  w.r.t.  $Mod$ .

Example

$$upd(AD, \{slaw(w), head(w, l_1), prec(w, 1, l_2)\}) =$$

$$AD \cup \{ l_1 \text{ if } l_2 \}$$

- *Symptom*:  $H^{cT}$  such that  $\alpha(\langle AD, H^{cT} \rangle)$  is inconsistent.
- *Modification* of  $AD$  for symptom  $H^{cT}$ : valid  $Mod$  such that  $\alpha(\langle upd(AD, Mod), H^{cT} \rangle)$  is consistent
- *Candidate Correction*:  $cC = \langle Mod, \Delta \rangle$ , where:
  - ◊  $Mod$ : modification of  $AD$  for  $H^{cT}$
  - ◊  $\Delta$ : components that may be damaged by actions of  $H^{cT}$  according to  $upd(AD, Mod)$ .

# Learning

- Approach: generate and test.
- *Generation*: sets of possible valid modification statements are generated.
- *Testing*: *Mod*'s that do not allow to explain the observations are discarded.

# Learning Module

Consists of  $\alpha(\langle AD, H^{cT} \rangle)$  together with:

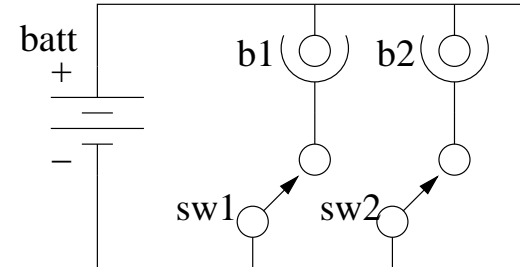
*CGEN* :

- { % Any *Lit* can be a precondition of a law  
 $\{ prec(W, N, Lit) \} \leftarrow law(W).$
- { % Available law names can be used for new laws  
 $\{ new\_law(W) : avail\_law\_name(W) \}.$
- { % New laws are either state constr's or dynamic laws  
 $1\{ dlaw(W), slaw(W) \} 1 \leftarrow new\_law(W).$
- { % Any *Lit* can be the head of a new law  
 $1\{ head(W, Lit) \} 1 \leftarrow new\_law(W).$
- { % Any action *Act* can be the trigger of a new dynamic law  
 $1\{ action(W, Act) \} 1 \leftarrow new\_law(W), dlaw(W).$
- {  $\leftarrow holds(F, T), obs(\neg F, T).$
- {  $\leftarrow holds(\neg F, T), obs(F, T).$



## Example: Learning about the Circuit

- $H^{cT}$ : 
$$\begin{cases} obs(closed(sw_1), 0), \quad obs(\neg closed(sw_2), 0), \\ obs(lit(b_1), 0), \quad obs(\neg lit(b_2), 0) \\ hpd(flip(sw_2), 0) \\ obs(\neg lit(b_2), 1) \end{cases}$$



- $\alpha(\langle AD, H^{cT} \rangle)$  inconsistent  $\Rightarrow H^{cT}$  is *symptom*
- Generation: possible set of modification statements is:
  - $slaw(w_0),$
  - $head(w_0, ab(batt)),$
  - $prec(w_0, 1, closed(sw_1)), \quad prec(w_0, 2, closed(sw_2))$
- Testing: according to the (new) model,  $obs(\neg lit(b_2), 1)$  is justified.

**CANDIDATE CORRECTION FOUND!!**

## About the Complete Architecture

- Diagnostic and learning modules *gather further observations* to confirm their hypotheses.
- Extension of A-Prolog (*CR-Prolog*) allows computing:
  - plans that satisfy *at best* a set of requirements
  - *most likely* diagnoses
  - *most reasonable* corrections
- *More powerful encoding of  $\mathcal{AL}$  in A-Prolog* allows learning of more general laws, e.g:

$$ab(batt) \quad \text{if} \quad \begin{array}{l} closed(SW_1), \\ closed(SW_2), \\ SW_1 \neq SW_2. \end{array}$$

# Conclusions

Unique features:

- The architecture uniformly combines planning, diagnosis and learning.
- Sophisticated reasoning + use of observations  $\Rightarrow$  high degree of autonomy.
- Shared domain model  $\Rightarrow$  ease of development, verification, maintenance.
- Directly implementable.

## A-Prolog Standpoint

- Demonstration of the flexibility of A-Prolog.  
A-Prolog can be used for:
  - ◇ Axiomatizing models and histories.
  - ◇ Encoding general purpose reasoning modules.
  - ◇ Formalizing control knowledge, i.e. constraints and preferences.
  - ◇ High level specification and direct implementation.

## Future Work

- Planning with incomplete information: possible plans and their relation with sensing.
- Automated goal selection (CR-Prolog's preferences?).
- Continue work on confirmation of hypotheses in presence of non-observable fluents.
- Introduce confirmation of hypotheses as a subgoal in the observe-think-act loop.