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: \( \text{lit}(b_1) \)

- **Observes:** switches open; bulbs off; components ok
- **Finds plan:** \( \text{flip}(sw_1) \)
- **Executes:** \( \text{flip}(sw_1) \)
- **Observes:** ...?
Diagnosis

[...]

- **Executes**: \(\text{flip}(sw_1)\)

- **Observes**: \(\neg \text{lit}(b_1)\) \(\Rightarrow\) **UNEXPECTED!!!**

- **Explains**: \(\text{blow\_up}(b_1)\) occurred *concurrently* with \(\text{flip}(sw_1)\)

- **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) \iff SUCCESS!!!
Beyond Diagnosis
What if...?

Receives new goal: $\text{lit}(b_2)$

- Finds plan: $\text{flip}(sw_2)$
- Executes: $\text{flip}(sw_2)$
- Observes: $\neg \text{lit}(b_2) \iff \text{UNEXPECTED!!!}$
- Explains: $\text{blow up}(b_2)$ occurred (e.g. with $\text{flip}(sw_2)$)
- Tests: is $ab(b_2)$ true?

- Answer: $ab(b_2)$ false!!! $\iff$ 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**: \( lit(b_2) \) ⇐ **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 \textit{(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:

\[
\begin{align*}
&\text{obs}(\neg \text{closed}(sw_1), 0), \; \text{obs}(\neg \text{closed}(sw_2), 0), \\
&\text{obs}(\neg \text{lit}(b_1), 0), \; \text{obs}(\neg \text{lit}(b_2), 0), \\
&\text{obs}(\neg \text{ab}(b_1), 0), \; \text{obs}(\neg \text{ab}(b_2), 0), \\
&\text{obs}(\neg \text{ab}(\text{batt}), 0),
\end{align*}
\]

Agent Actions at step 0:

\[
\begin{align*}
&\text{hpd}(\text{flip}(sw_1), 0),
\end{align*}
\]

Observations at step 1:

\[
\begin{align*}
&\text{obs}(\text{closed}(sw_1), 1), \; \text{obs}(\neg \text{lit}(b_1), 1)
\end{align*}
\]
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.”

\[ \text{take_umbrella} \leftarrow \text{raining}, \neg \text{have_raincoat}. \]

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

\[ \text{raining}, \neg \text{have_raincoat}. \]

**Conclusion:** \[ \text{take_umbrella} \leftarrow \text{the agent take an umbrella}. \]
Simple Examples

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

\[ \neg \text{have}_B \leftarrow \text{good}_\text{student}. \]

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

\[ \text{have}_B. \ \text{good}_\text{student}. \]

**Contradiction:** conclusion \( \neg \text{have}_B \) contradicts \( \text{have}_B \).

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.”

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

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

$$\text{closed}(sw_1). \text{closed}(sw_2). \text{closed}(sw_3).$$

$$\text{connected}(sw_1, b_1). \text{connected}(sw_2, b_2). \text{connected}(sw_3, b_3).$$

$$\neg \text{ab}(b_1). \text{ab}(b_2). \neg \text{ab}(b_3).$$

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

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

\[ \{X \mid \text{have}(X)\} \subseteq \{X \mid \text{toy}(X)\} \leftarrow \text{behave}. \]

Given facts: \textit{behave}, \textit{toy}(t_1), \textit{toy}(t_2)

Answer Sets:

\[ \{\text{have}(t_1), \text{have}(t_2)\} \]
\[ \{\text{have}(t_1)\} \]
\[ \{\text{have}(t_2)\} \]
\[ \{\} \leftarrow \text{agent does not get any toys} \]

Abbreviation: \{\textit{have}(X) : \textit{toy}(X)\} \leftarrow \textit{behave}.\]
**Translation of $AD$ in A-Prolog, $\alpha(AD)$**

Dynamic Law of $\mathcal{AL}$

\[ d_1 : \text{flip}(sw_1) \text{ causes } \text{closed}(sw_1) \text{ if } \neg \text{closed}(sw_1). \]

$\alpha$-Translation:

\[
\begin{align*}
\text{\% } d_1 \text{ is a dynamic law} \\
\text{dlaw}(d_1).
\end{align*}
\]

\[
\begin{align*}
\text{\% The head of } d_1 \text{ is } \text{closed}(sw_1) \\
\text{head}(d_1, \text{closed}(sw_1)).
\end{align*}
\]

\[
\begin{align*}
\text{\% The action of } d_1 \text{ is } \text{flip}(sw_1) \\
\text{action}(d_1, \text{flip}(sw_1)).
\end{align*}
\]

\[
\begin{align*}
\text{\% Precondition \#1 of } d_1 \text{ is } \neg \text{closed}(sw_1) \\
\text{prec}(d_1, 1, \neg \text{closed}(sw_1)).
\end{align*}
\]
Translating State Constraints

Law:

\[ \text{lit}(b_1) \text{ if closed}(sw_1), \neg \text{ab}(b_1) \]

\(\alpha\)-Translation:

\[
\begin{align*}
\% \ s_1 \text{ is a state constraint} \\
\text{slaw}(s_1).
\end{align*}
\]

\[
\begin{align*}
\% \text{ The head of } s_1 \text{ is } \text{lit}(b_1) \\
\text{head}(s_1, \text{lit}(b_1)).
\end{align*}
\]

\[
\begin{align*}
\% \text{ The preconditions of } s_1 \text{ are } \text{closed}(sw_1) \text{ and } \neg \text{ab}(b_1) \\
\text{prec}(s_1, 1, \text{closed}(sw_1)). \\
\text{prec}(s_1, 2, \neg \text{ab}(b_1)).
\end{align*}
\]
Projecting the Effects of Actions

\[ holds(L, T + 1) \leftarrow \text{dlaw}(D), \]
\[ \text{head}(D, L), \]
\[ \text{action}(D, A), \]
\[ \text{occurs}(A, T), \]
\[ \text{all}_{\text{prec}}_{\text{hold}}(D, T). \]

\[ \ldots \]

\[ \text{prec}_{\text{n}}_{\text{holds}}(D, N, T) \leftarrow \text{prec}(D, N, P), \]
\[ \text{holds}(P, T). \]
Planning
Overview

• Agent’s Goal: set of fluent literals, e.g.
  \[
  \{ \text{have(lots_of(money))}, \, \neg \text{in(jail)} \}. 
  \]

• Approach: generate and test.

• \textit{Generation}: possible occurrences of actions are generated.

• \textit{Testing}: constraint ensuring that solutions achieve the goal.
Planning Module

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

$$
\text{PGEN :} \begin{cases} 
\text{%%% select occurrences of actions for each step} \\
\{ \text{occurs}(A, T) : \text{ag\_action}(A) \} \leftarrow T \geq cT.
\end{cases}
$$

$$
\text{%%% goal achieved if required literals eventually hold} \\
\text{goal\_achieved} \leftarrow \text{holds}(g_1, T), \\
\quad \ldots, \\
\quad \text{holds}(g_m, T).
$$

$$
\text{%%% plans achieve the goal} \\
\leftarrow \text{not goal\_achieved}.
$$
Example

$H^c T$: \[
\begin{align*}
\text{obs}(\neg \text{closed}(sw_1), 0), & \quad \text{obs}(\neg \text{closed}(sw_2), 0), \\
\text{obs}(\neg \text{lit}(b_1), 0), & \quad \text{obs}(\neg \text{lit}(b_2), 0), \\
\text{obs}(\neg \text{ab}(b_1), 0), & \quad \text{obs}(\neg \text{ab}(b_2), 0), \\
\text{obs}(\neg \text{ab}(\text{batt}), 0)
\end{align*}
\]

- Goal: \{\text{lit}(b_1)\}.
- Generation: possible sequence of actions is:

$$\text{occurs}(\text{flip}(sw_1), 0).$$

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

$$\text{holds}(\text{lit}(b_1), 1).$$

**PLAN FOUND!!**
Diagnosis
Basics

• **Symptom:** history $H_c^T$ with unexpected observations

• $H_c^T$ is symptom if:

  $$\alpha(\langle AD, H_c^T \rangle) \text{ is inconsistent}$$

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

  $$\alpha(\langle AD, H_c^T \cup E \rangle) \text{ is consistent}.$$ 

• **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
\{ occurs($A, T$) : ex_action($A$) \} $\leftarrow$ $0 \leq T < cT$.

%% It is impossible for a prediction to
%% disagree with an observation.
$\leftarrow$ holds($F, T$), $\text{obs}(\neg F, T)$.
$\leftarrow$ holds($\neg F, T$), $\text{obs}(F, T)$. 
Example: Diagnosing the Circuit

\[ H^{cT}: \begin{cases} 
  \text{obs}(\neg \text{closed}(sw_1), 0), \ \text{obs}(\neg \text{closed}(sw_2), 0), \\
  \text{obs}(\neg \text{lit}(b_1), 0), \ \text{obs}(\neg \text{lit}(b_2), 0), \\
  \text{obs}(\neg \text{ab}(b_1), 0), \ \text{obs}(\neg \text{ab}(b_2), 0), \ \text{obs}(\neg \text{ab}(\text{batt}), 0) \\
  \text{hpd}(flip(sw_1), 0) \\
  \text{obs}(\neg \text{lit}(b_1), 1) 
\end{cases} \]

\[ \alpha(\langle AD, H^{cT} \rangle) \text{ inconsistent } \Rightarrow H^{cT} \text{ is symptom} \]

• Generation: possible sequence of actions is:

\[ \text{occurs}(\text{blow}_\uparrow \text{up}(b_1), 0) \]

• Testing: according to the model, \( \text{occurs}(\text{blow}_\uparrow \text{up}(b_1), 0) \) justifies:

\[ \text{obs}(\neg \text{lit}(b_1), 1). \]

CANDIDATE DIAGNOSIS FOUND!!
Learning
Modification Statements

- **Modification Statements**: $dlaw(w)$, $slaw(w)$, $head(w,l)$, $action(w,a_e)$, $prec(w,n,p)$.

- **Valid** set of Modification Statements, $Mod$:
  - $\diamond$ for every $w$, $slaw(w)$ and $dlaw(w)$ cannot be both in $Mod$;
  - $\diamond$ one $head(w,l)$ statement for every $slaw(w)$ or $dlaw(w)$ in $Mod$;
  - $\diamond$ one $action(w,l)$ statement for every $dlaw(w)$ in $Mod$;

**Examples**

- $\{slaw(w), head(w,l_1), prec(w,1,l_2)\}$ is valid;
- $\{slaw(w), prec(w,1,l_2)\}$ is not valid (missing $head$);
- $\{dlaw(w), head(w,l_1), prec(w,1,l_2)\}$ is not valid (missing $action$).
Candidate Correction

- ** upd**(AD, Mod): Update of AD w.r.t. Mod.
  
  Example
  
  upd**(AD, {slaw(w), head(w, l1), prec(w, 1, l2)}) =
  
  AD ⋃ { l1 if l2 }

- **Symptom**: HcT such that \( \alpha(\langle AD, HcT \rangle) \) is inconsistent.

- **Modification** of AD for symptom HcT: valid Mod such that
  
  \( \alpha(\langle upd(AD, Mod), HcT \rangle) \) is consistent

- **Candidate Correction**: cC = ⟨Mod, ∆⟩, where:
  
  ◇ Mod: modification of AD for HcT
  
  ◇ ∆: components that may be damaged by actions of HcT 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:

$$\begin{align*}
\text{CGEN :} & \quad \\
\% \text{ Any Lit can be a precondition of a law} \\
\{ \text{prec}(W, N, \text{Lit}) \} & \leftarrow \text{law}(W).
\%
\% \text{ Available law names can be used for new laws} \\
\{ \text{new_law}(W) : \text{avail_law_name}(W) \}.
\%
\% \text{ New laws are either state constr’s or dynamic laws} \\
1\{ \text{dlaw}(W), \text{slaw}(W) \} & 1 \leftarrow \text{new_law}(W).
\%
\% \text{ Any Lit can be the head of a new law} \\
1\{ \text{head}(W, \text{Lit}) \} & 1 \leftarrow \text{new_law}(W).
\%
\% \text{ Any action Act can be the trigger of a new dynamic law} \\
1\{ \text{action}(W, \text{Act}) \} & 1 \leftarrow \text{new_law}(W), \text{dlaw}(W).
\%
\leftarrow \text{holds}(F, T), \text{obs}(\neg F, T).
\leftarrow \text{holds}(\neg F, T), \text{obs}(F, T).
\end{align*}$$
Example: Learning about the Circuit

- $H^{cT}$:
  \[
  \begin{align*}
  \text{obs}(\text{closed}(sw_1), 0), \quad & \text{obs}(\neg \text{closed}(sw_2), 0), \\
  \text{obs}(\text{lit}(b_1), 0), \quad & \text{obs}(\neg \text{lit}(b_2), 0) \\
  \text{hpd}(\text{flip}(sw_2), 0) \\
  \text{obs}(\neg \text{lit}(b_2), 1)
  \end{align*}
  \]

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

- Generation: possible set of modification statements is:
  \[
  \begin{align*}
  \text{slaw}(w_0), \\
  \text{head}(w_0, ab(batt)), \\
  \text{prec}(w_0, 1, \text{closed}(sw_1)), \quad & \text{prec}(w_0, 2, \text{closed}(sw_2))
  \end{align*}
  \]

- Testing: according to the (new) model, $\text{obs}(\neg \text{lit}(b_2), 1)$ 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 AL in A-Prolog* allows learning of more general laws, e.g:

\[
ab(batt) \text{ if } \begin{align*}
closed(SW_1), \\
closed(SW_2), \\
SW_1 \neq SW_2.
\end{align*}
\]
Conclusions

Unique features:

- The architecture uniformly combines planning, diagnosis and learning.

- Sophisticated reasoning + use of observations ⇒ high degree of autonomy.

- Shared domain model ⇒ ease of development, verification, maintainance.

- 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.