Answer Set Based Design of Autonomous, Rational Agents

Marcello Balduccini

Knowledge Representation Lab
Computer Science Department
Texas Tech University

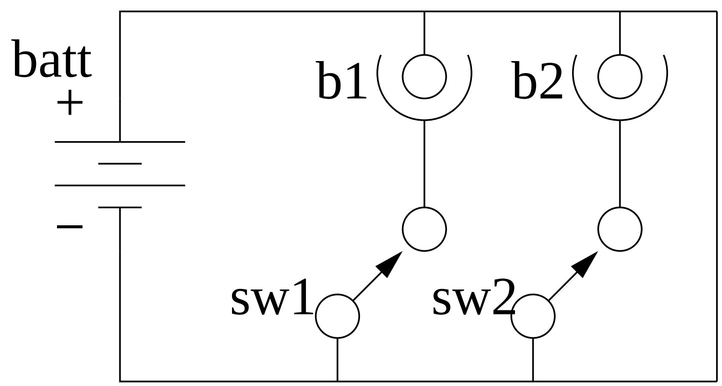
October 7, 2005

Introduction

- **Goal (1)**: design an agent capable of rational, autonomous interaction.
- **Goal (2)**: all reasoning modules written in A-Prolog and sharing the same domain model.
  - ◇ unique model: ease of development and maintenance.
  - ◇ all reasoning in A-Prolog: demonstrates the power of the language.
- **Why A-Prolog**:
  - ◇ high-level specification language, *but also...*
  - ◇ ...close to implementation level;
  - ◇ reasoning modules are compact and easy to understand.

Desired Agent Behavior

A Physical System



Fluents

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

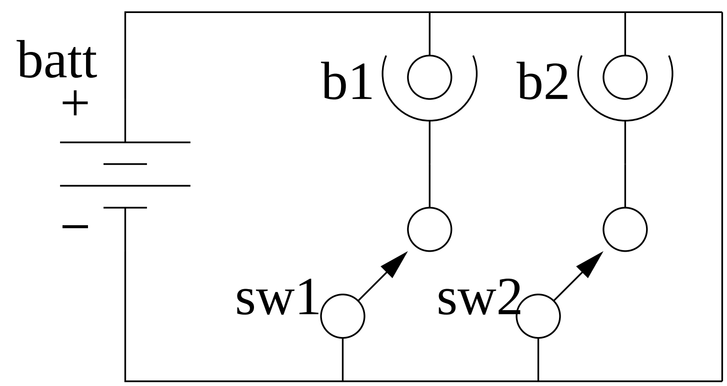
Agent Actions

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

Exogenous Actions

- $blow\_up(Bulb)$

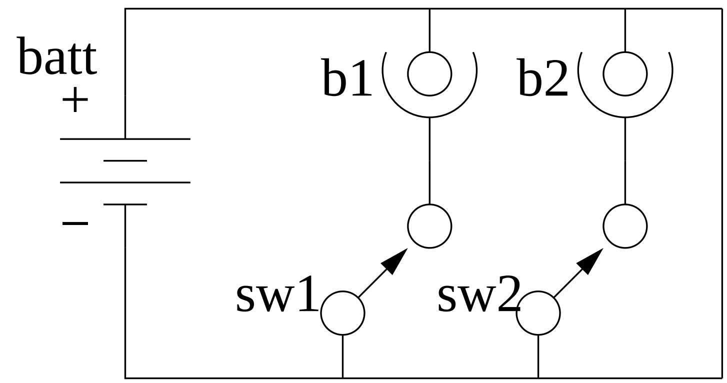
Planning



Agent's goal: $lit(b_1)$

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

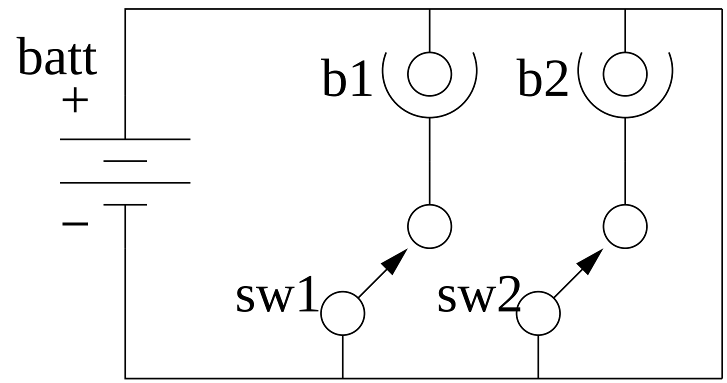
Diagnosis



[. . .]

- **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

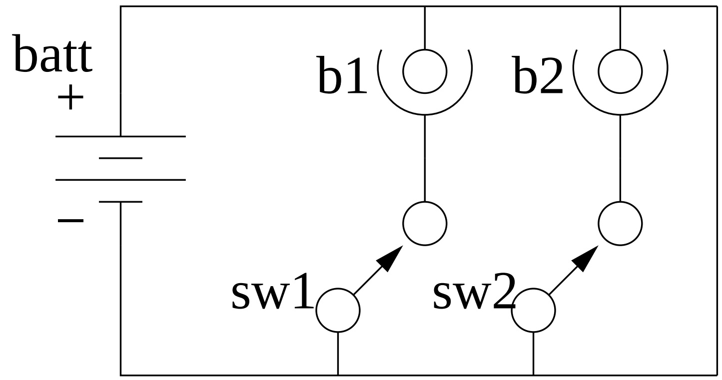


[. . .]

- **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!!!**

Another Type of Reasoning

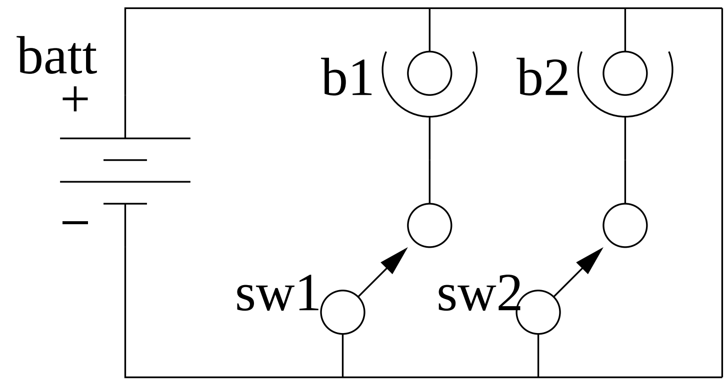
What if...?



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**!!!

Learning

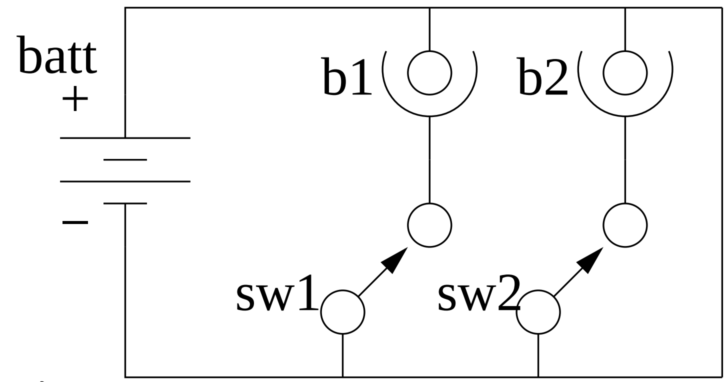


[...]

- **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**: $lit(sw_2)$ $\Leftarrow$ **SUCCESS!!!**

The Agent Control Loop

Key Elements

Reasoning Processes

- Planning
- Interpreting observations 1: diagnosis
- Interpreting observations 2: (inductive) learning
- Testing explanations

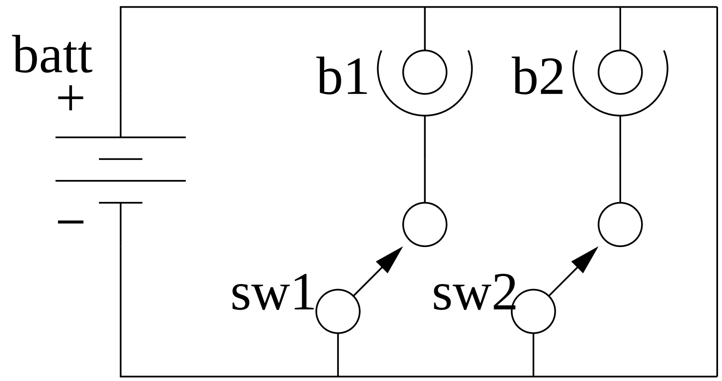
Other Processes

- Observation gathering
- Execution of actions

Observe-Think-Act loop

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

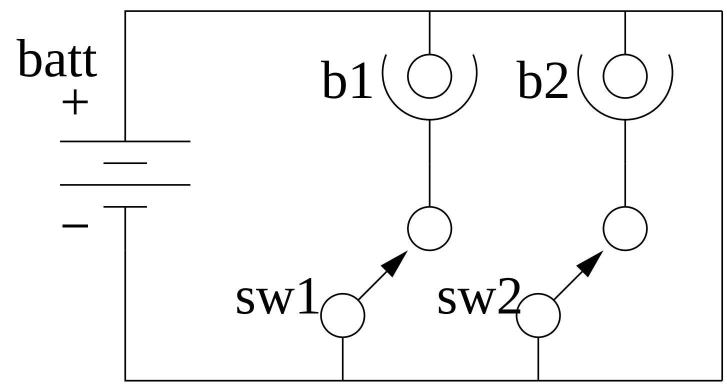
Agent Behavior Revisited



Agent's goal: $lit(b_1)$

- **Observes**: switches open; bulbs off; ... ⇐ **STEP** 1
- *no interpretation needed* ⇐ **STEP** 2
- *goal given by the user* ⇐ **STEP** 3
- **Finds plan**: $flip(sw_1)$ ⇐ **STEP** 4

Agent Behavior Revisited



[. . .]

- **Executes**: $flip(sw_1)$ ⇐ **STEP 5**
- **Observes**: $\neg lit(b_1)$ ⇐ **STEP 1**
- **Diagnosis**: $blow\_up(b_1)$ occurred ⇐ **STEP 2**
- **Tests**: is $ab(b_1)$ true? ⇐ **STEP 2**

Overall Choices

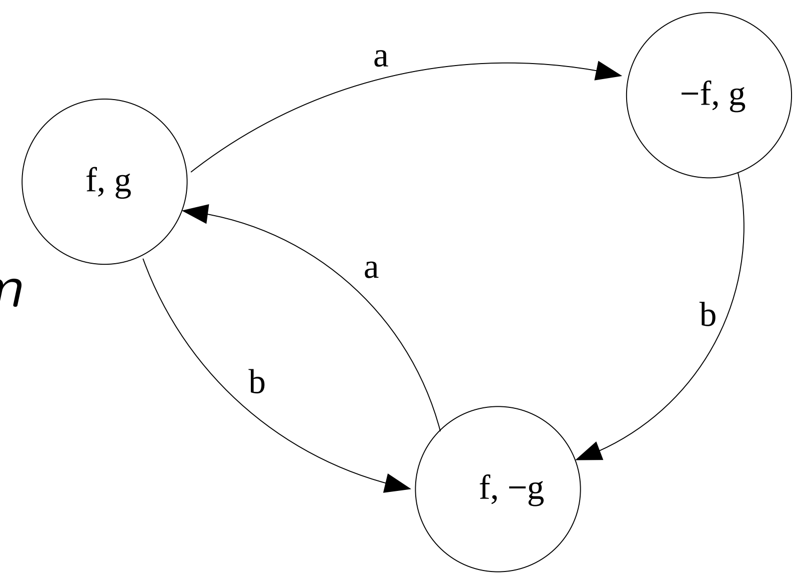
- I/O, link among the reasoning modules: *procedural code*
- Reasoning processes: *answer set programming*
  - ◇ Domain model axiomatized in $\mathcal{AL}$
  - ◇ Reasoning reduced to finding answer sets
  - ◇ Reasoning modules written in A-Prolog/CR-Prolog

Domain Axiomatization

Language $\mathcal{AL}$

- $\mathcal{AL}$ is an *action language*

- Central concept: *transition diagram*



- Divided in:
  - ◇ $\mathcal{AL}_d$: describes *effects of actions*;
  - ◇ $\mathcal{AL}_h$: describes *recorded history* of the domain;
  - ◇ $\mathcal{AL}_q$: *query language*.

Syntax of $\mathcal{AL}_d$

- Dynamic Laws:

$$a \text{ causes } p \text{ if } q,\ \neg r.$$

- State Constraints:

$$p \text{ if } q,\ \neg r.$$

- Executability Conditions:

$$a \text{ impossible\_if } p,\ \neg q.$$

- Action Description (AD): set of laws of the types above

Semantics of $\mathcal{AL}_d$

- Defined by *transition diagram*, $Trans(AD)$.
- The core is the *successor state axiom*.

Given:

- ◇ states σ, σ';
- ◇ action a executable in σ;
- ◇ Z: set of all state constraints from action description.

Successor State Axiom:

$$\sigma' = Cn_Z(E(a,\sigma) \cup (\sigma \cap \sigma'))$$

Language $\mathcal{AL}_h$

Syntax

- $obs(Literal, Step)$: $Literal$ observed to hold at step $Step$;
- $hpd(Action, Step)$: $Action$ observed to happen at step $Step$;
- Recorded History: $\langle H, cT \rangle$, where
  - ◇ H: set of $\mathcal{AL}_h$ statements;
  - ◇ cT: current time step.
- $\langle H, cT \rangle$ *is also written* H^{cT}.

Semantics

- Model of H^{cT} w.r.t. AD:
  trajectory in $Trans(AD)$ *matching* H^{cT}.

Domain Descriptions of $\mathcal{AL}$

- Domain Description (DD): $\langle AD, H^{cT} \rangle$
- DD is *translated to A-Prolog* for actual computation

$\alpha(DD)$ is the A-Prolog translation of DD

$\alpha(DD)$ includes the *Reality Axioms*:

% L observed at 0 $\Rightarrow$ holds at 0 in every model of DD.
$h(L, 0) \leftarrow obs(L, 0).$

% A observed at T $\Rightarrow$ occurs at T in every model of DD.
$o(A, T) \leftarrow hpd(A, T).$

% It is impossible for a state of a model of DD not to
% match the observations.
$\leftarrow obs(L, T), \text{not } h(L, T).$

Action Description for the Example

%% Flipping SW causes SW to become
%% closed if it was open and vice-versa.
%%
$flip(SW)$ causes $closed(SW)$ if $\neg closed(SW)$.
$flip(SW)$ causes $\neg closed(SW)$ if $closed(SW)$.

$lit(b_1)$ if $closed(sw_1),\ \neg ab(b_1)$.
$[\ldots]$

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

$replace(batt)$ causes $\neg ab(batt)$.
$[\ldots]$

α-Translation of State Constraints

Law:

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

α-**Translation**:

$$\left\{\begin{array}{l}
\%\ s_1 \text{ is a state constraint} \\
slaw(s_1). \\
\\
\%\ \text{the } \textit{head} \text{ of } s_1 \text{ is } lit(b_1) \\
head(s_1, lit(b_1)). \\
\\
\%\ \text{the } \textit{preconditions} \text{ of } s_1 \text{ are } closed(sw_1) \text{ and } ab(b_1) \\
prec(s_1, closed(sw_1)). \\
prec(s_1, \neg ab(b_1)).
\end{array}\right.$$

α-Translation of Dynamic Laws

Law:

$$flip(sw_1) \text{ causes } closed(sw_1) \text{ if } \neg closed(sw_1).$$

α-**Translation**:

% d_1 is a dynamic law with head $closed(sw_1)$
$dlaw(d_1).$
$head(d_1, closed(sw_1)).$

% the *trigger* of d_1 is $flip(sw_1)$
$trigger(d_1, flip(sw_1)).$

% the *precondition* of d_1 is $closed(sw_1)$
$prec(d_1, closed(sw_1)).$

Planning

A-Prolog Planning Module

Finds plans of length up to k, given:

- goal $\{g_1, \ldots, g_m\}$ (set of literals)
- H^{cT}

$$PGEN(k) : \begin{cases} \text{\%\% select at least one action per step} \\ 1\{o(A,T) : ag\_action(A)\} \leftarrow cT \leq T < cT + k. \\ \\ \text{\%\% goal achieved if required literals eventually hold} \\ goal\_achieved \leftarrow h(g_1, cT + k), \\ \qquad\qquad\qquad\qquad \ldots, \\ \qquad\qquad\qquad\qquad h(g_m, cT + k). \\ \\ \text{\%\% plans achieve the goal} \\ \leftarrow \text{not } goal\_achieved. \end{cases}$$

Shortest Plan Algorithm

Input:

- domain description, $DD = \langle AD, H^{cT} \rangle$;
- goal $g = \{l_1, \ldots, l_m\}$.

Output:

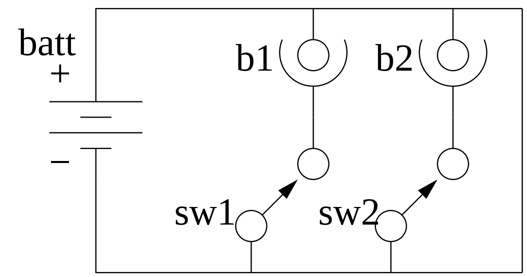
- a shortest plan for g.

Steps:

1. k := 0
2. **if** $\alpha(DD) \cup PGEN(k)$ is consistent, **then**
3. extract the plan from one answer set and **return** it
4. k := k + 1
5. **goto** 2

Example: Planning in the Circuit

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



- Goal: $\{lit(b_1)\}$
- With $k = 0$:

empty sequence of actions; $h(lit(b_1), 0)$ does not hold

$$\downarrow$$

$\alpha(DD) \cup PGEN(0)$ is *inconsistent*.

- With $k = 1$:

$o(flip(sw_1), 0)$ can be selected; if so, $h(lit(sw_1), 1)$ holds

$$\downarrow$$

$\alpha(DD) \cup PGEN(1)$ is *consistent*.

Diagnosis

Unexpected Observations: Symptoms

- *Symptom*: history H^{cT} with unexpected observations
- Checking if H^{cT} is symptom $\rightarrow$ testing consistency of:

$$\alpha(DD) \cup R$$

where

$$R : \begin{cases} \text{\%\% Awareness axioms: every fluent } F \text{ is} \\ \text{\%\% initially either true or false} \\ \text{\%} \\ h(F, 0) \leftarrow \text{not } h(\neg F, 0). \\ h(\neg F, 0) \leftarrow \text{not } h(F, 0). \end{cases}$$

Candidate Diagnoses

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

$$\alpha(DD) \cup E \cup R \text{ is consistent.}$$

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

- Finding $cD \longrightarrow$ answer sets of:

$$D_0(DD) = \alpha(DD) \cup R \cup$$

$$\{o(A,T) : ex\_action(A)\} \leftarrow 0 \leq T < cT - 1.$$

Finding Diagnoses

- Agent needs to verify if components in Δ_E are faulty.

function $Find\_Diag$(**var** H^{cT}: **symptom**): diagnosis of H^{cT}

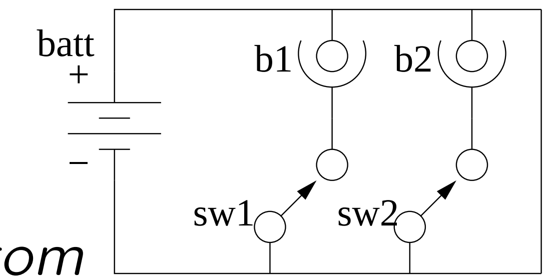
```
repeat
   ⟨E, Δ_E⟩ := Candidate_Diag(H^cT);
   if E = ∅ return ⟨E, Δ_E⟩; { no diagnosis could be found }
   diag := true;     Δ_0 = Δ_E
   while  Δ_0 ≠ ∅  and  diag  do
      selet c ∈ Δ_0;     Δ_0 := Δ_0 \ {c};
      if  observe(cT, ab(c)) = true
         then H^cT := H^cT ∪ obs(ab(c), cT);
         else H^cT := H^cT ∪ obs(¬ab(c), cT);   diag := false;
      end {if}
   end {while}
until diag;

return ⟨E, Δ_E⟩

end
```

Example: Diagnosing the Circuit

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



1. $\alpha(DD) \cup R$ inconsistent $\Rightarrow H^{cT}$ is *symptom*
2. Finding a candidate diagnosis:

$$o(blow\_up(b_1), 0) \text{ can be selected}$$
$$\downarrow$$
$$cD = \langle\{o(blow\_up(b_1), 0)\}, \{b_1\}\rangle$$

3. Testing: $observe(ab(b_1), cT)$ holds $\Rightarrow$ cD is diagnosis.

Learning

Candidate Corrections

- *Modification* of AD for symptom H^{cT}: AD' such that

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

- *Candidate Correction*: $cC = \langle AD', \Delta_{AD'} \rangle$, where:
  - ◇ AD': modification of AD for H^{cT}
  - ◇ $\Delta_{AD'}$: components that may be damaged by actions of H^{cT} according to AD'.

- Modifications considered:
  - addition of laws;
  - addition of possibly non-ground preconditions to the laws.

Conservative Modifications

- Modifications consisting of:
  - ◇ addition of laws;
  - ◇ addition of preconditions to the laws.

Conservative modifications for the Example:

- Add state constraint s:

$$ab(batt) \text{ if } \{\}. \quad \textit{(empty body)}$$

- Add preconditions $closed(sw_1), closed(sw_2)$ to the body of s:

$$ab(batt) \text{ if } closed(sw_1), closed(sw_2).$$

Only Conservative Modifications are considered here.

Computing Candidate Corrections

- Candidate Corrections of AD for H^{cT} $\rightarrow$ answer sets of:

$$L_0(H^{cT}) = \alpha(\langle AD, H^{cT} \rangle) \; \cup R \cup :$$

$$CGEN : \begin{cases} \text{\% Any } Lit \text{ can be a precondition of a law} \\ \{\ prec(W, Lit)\ \} \leftarrow law(W). \\ \\ \text{\% Available law names can be used for new laws} \\ \{\ new\_law(W) : \ avail\_law\_name(W)\ \}. \\ \\ \text{\% New laws are either state constr's or dynamic laws} \\ dlaw(W) \text{ or } slaw(W) \leftarrow new\_law(W). \\ \\ \text{\% Any } Lit \text{ can be the head of a new law} \\ 1\{\ head(W, Lit)\ \}1 \leftarrow new\_law(W). \\ \\ \text{\% Any action } Act \text{ can be the trigger of a new dynamic law} \\ 1\{\ trigger(W, Act)\ \}1 \leftarrow new\_law(W), dlaw(W). \end{cases}$$

Computing Corrections

function $Find\_Correction(AD$, **var** H^{cT}: **symptom**): a correction for H^{cT}

repeat
$\langle AD', \Delta \rangle := Candidate\_Correction(AD, H^{cT})$;

if $AD' = \emptyset$ **return** $\langle cAD, \Delta \rangle$ { no correction found }

corr_found := $true$; $\Delta_0 := \Delta$;

while $\Delta_0 \neq \emptyset$ **and** corr_found **do**
select $c \in \Delta_0$; $\Delta_0 := \Delta_0 \setminus \{c\}$;

if $observe(cT, ab(c)) = true$
then $O := O \cup obs(ab(c), cT)$;
else $O := O \cup obs(\neg ab(c), cT)$; corr_found := $false$;
end {if}

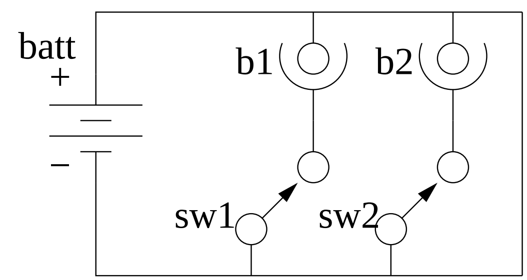
end {while}

until corr_found;

return $\langle AD', \Delta \rangle$
end

Example: Learning about the Circuit

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



1. $\alpha(DD) \cup R$ inconsistent $\Rightarrow H^{cT}$ is *symptom*
2. Finding a candidate correction:
   - ◇ Selection: $new\_law(w_0)$, $slaw(w_0)$,
     $head(w_0, ab(batt))$
     $prec(w_0, closed(sw_1))$, $prec(w_0, closed(sw_2))$
   - ◇ $\alpha(\langle AD', H^{cT}\rangle) \cup R \cup CGEN$ is consistent
3. Testing: $observe(ab(batt), cT)$ holds $\Rightarrow$ correction found.

More Complex Corrections

Non-ground state constraints.

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

Non-ground dynamic laws.

$$
\begin{array}{lllll}
touch(P, C_2) & \text{causes} & ab(C_1) & \text{if} & statically\_charged(P), \\
 & & & & connected(C_1, C_2).
\end{array}
$$