Lecture Nine: Modeling Dynamic Domains (II)

301315 Knowledge Representation and Reasoning

A General Solution

- We have known how to model a dynamic domain like the Blocks World using ASP.
- Basically, we create an ASP program, which contains two parts: the initial configuration and the domain laws.
- Now beyond the Blocks World, we are looking for a general solution for modeling dynamic domains.
- One possible way to model a dynamic system is by a **transition diagram** - a directed graph whose
  - nodes correspond to physically possible states of the domain
  - arcs are labeled by actions.

A General Solution - Transitions

A *transition* $< \sigma_0, \{a_1, \cdots, a_k\}, \sigma_1 >$ of the diagram, where $\{a_1, \cdots, a_k\}$ is a set of action executable in state σ_0, indicates that σ_1 may be a result of simultaneous execution of these actions in σ_0.

Our representation guarantees that the effect of an action depends only on the state in which that action was executed. The way in which this state was reached is irrelevant.

A General Solution - Transition Diagrams

- A *path* $<\sigma_0, a_0, \sigma_1, \cdots, a_{n-1}, \sigma_n>$ of the diagram represents a possible *trajectory* of the system with initial state σ_0 and final state σ_n.
- The transition diagram for a system contains all possible trajectories of that system.
- Such a transition diagram can be quite complicated.
- Things get even more complicated because of the need to specify *what is not changed by actions*.

A General Solution - The Frame Problem

- The **frame problem** (first time addressed by John McCarthy in 1960s) is the problem of finding a concise and accurate representation in a formal language of what is not changed by actions.
- Notice that in our Blocks World problem, we showed a solution to the Frame Problem. We simply reduced it to finding a representation of the *Inertia Axiom* - a default that states that things normally stay as they are.

A General Solution - Causal Laws

- A **causal law** represents the effect of an action, which can be defined as follows:

  a **causes** f **if** $p_0, \cdots, p_m$.

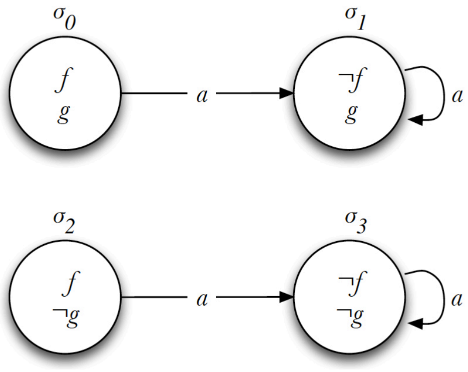
- The law says that action a, executed in a state which satisfies conditions $p_0, \cdots, p_m$, causes fluent f to become true in the resulting state.
- We have seen the use of such laws in our Blocks World example when we defined the effect of action `put(Block,Location)`.

A General Solution - Causal Laws

Example

Transition Diagram for a Causal Law:

a **causes** $\neg f$ *if* f.



A General Solution - State Constraints

State constraints are defined as follows, can be seen as indirect effect of actions:

$$f \ \ if \ p_0, \cdots, p_m$$

which say that every state satisfying conditions $p_0, \cdots, p_m$ must be also satisfy f.

- Finding concise ways for defining these effects is called the **Ramification Problem**.
- Together with the Frame Problem discussed above, the Ramification Problem caused substantial difficulties for researchers in their attempts to precisely define transitions of discrete dynamic systems.

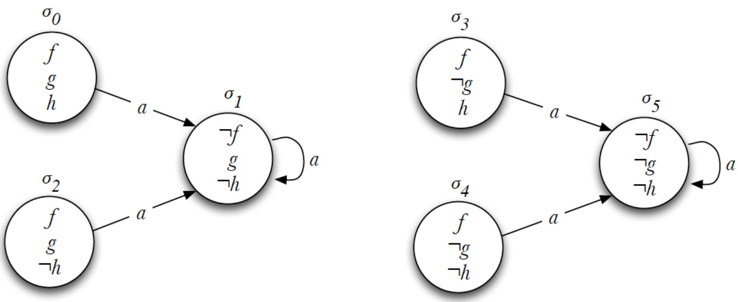
A General Solution - State Constraints

Example

Transition Diagram Including a State Constraint.

a **causes** $\neg f$ *if* f.

$\neg h$ **if** $\neg f$.



A General Solution - Executability Conditions

Executability conditions are represented by laws of the form

impossible $a_1, \cdots, a_k$ **if** $p_0, \cdots, p_m$

which say that it is impossible to execute actions $a_1, \cdots, a_k$ simultaneously in a state satisfying conditions $p_0, \cdots, p_m$.
As an example, let's add rule

impossible a **if** $\neg f$

which says that it is impossible to perform action a in any state which contains fluent $\neg f$.

A General Solution - Executability Conditions

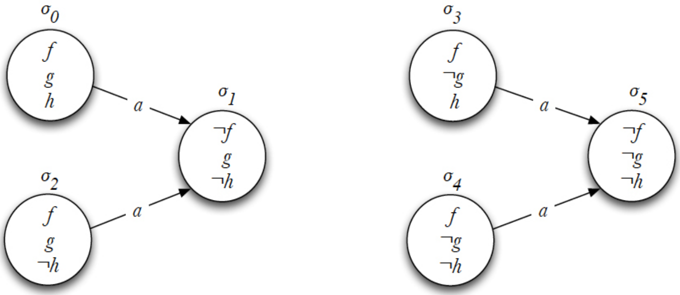
Example

Transition Diagram Including an Executability Condition.

a **causes** $\neg f$ **if** f

$\neg h$ **if** $\neg f$

impossible a **if** $\neg f$



Action Language $\mathcal{AL}$ - Syntax

- Action languages are formal models of parts of natural language used for describing the behaviour of dynamic systems.
- Looking at it another way, they are tools for describing transition diagrams.

Action Language $\mathcal{AL}$ - Syntax

- Action language $\mathcal{AL}$ is parametrized by a sorted signature containing three special sorts:
  - statics,
  - fluents, and
  - actions.
- The fluents are partitioned into two sorts: *inertial* and *defined*.
- **domain properties**: statics and fluents
- **domain literal** - domain property or its negation. We can have **fluent literals** and **static literals**.

Action Language $\mathcal{AL}$ - Syntax

- A set S of domain literals is called **complete** if for any domain property p either p or $\neg p$ is in S.
- S is called **consistent** if there is no p such that $p \in S$ and $\neg p \in S$.

Action Language $\mathcal{AL}$ - Syntax

Language $\mathcal{AL}$ allows the following types of *statements*:

- Causal Laws:

  a **causes** l_{in} **if** $p_0, \cdots, p_m$

- State Constraints:

  l **if** $p_0, \cdots, p_m$

- Executability Conditions:

  impossible $a_1, \cdots, a_k$ **if** $p_0, \cdots, p_m$

where a is an action, l is an arbitrary domain literal, l_{in} is a literal formed by an inertial fluent, $p_0, \cdots, p_m$ domain literals, $k \geq 0$, and $m \geq 0$. Moreover, no negation of a defined fluent can occur in the heads of state constraints.

Action Language $\mathcal{AL}$ - Syntax

Definition

A **system description** of $\mathcal{AL}$ is a collection of statements of $\mathcal{AL}$.

Action Language $\mathcal{AL}$ - Semantics

- A system description serves as a specification of a transition diagram of a dynamic system.
- It describes all possible trajectories of the system.
- So, to define the meaning of $\mathcal{AL}$ statements, we need to define the states and legal transitions of the diagram.

Action Language $\mathcal{AL}$ - Semantics

Defining States

- If we didn't have to worry about defined fluents, we could define states as

  a complete and consistent set of domain literals satisfying the system's state constraints

- Unfortunately, this is not good enough to give defined fluents their intended meaning.

Action Language $\mathcal{AL}$ - Semantics

Example

The Problem with Defined Fluents.

Suppose we have two state constraints:

h **if** f
h **if** $\neg g$

where h is a defined fluent and f and g are inertial, i.e., can only be changed by actions.

Action Language $\mathcal{AL}$ - Semantics

h **if** f
h **if** $\neg g$

Consider the following candidate states:
$\sigma_0 = \{f, g, h\} \quad \sigma_1 = \{f, g, \neg h\} \quad \sigma_2 = \{\neg f, g, h\}$
Clearly, σ_0 is a state, σ_1 is not. Why?

How about σ_2?
Notice that σ_2 doesn't seem to satisfy the definition of h since the truth of h doesn't follow from any of its defining rules. However, it is complete and consistent.

Action Language $\mathcal{AL}$ - The Solution

- To capture the intended meaning of defined fluents, we turn to ASP.
- To see whether a candidate set is a state, we create an ASP program by:
  (a) Taking all the state constraints and replacing the **if** in them by a "$\leftarrow$".
  (b) Adding the CWA for each defined fluent.
  (c) Adding all the statics and inertial fluents from the candidate set to the program as facts.
  (d) Adding relevant inertial rules.

Definition

A complete and consistent set of domain literals is a **state** of the transition diagram defined by a system description if it is the unique answer set of the program thus created.

Action Language $\mathcal{AL}$ - Transition

- The definition of the transition relation of a diagram is also based on the notion of the answer set of a logic program.
- To describe a transition $< \sigma_0, a, \sigma_1 >$, we construct a program $\Pi(\mathcal{SD}, \sigma_0, a)$ consisting of
  (a) logic programming encodings of system description $\mathcal{SD}$,
  (b) initial state σ_0, and
  (c) set of action a,

  such that answer sets of this program determine the states where the system can move into after the execution of a in σ_0.

Translating $\mathcal{AL}$ into ASP - The General Idea

Recall that given a system description $\mathcal{SD}$ and a transition $< \sigma_0, a, \sigma_1 >$, where $a = \{a_0, \cdots, a_k\}$, we construct an ASP program $\Pi(\mathcal{SD}, \sigma_0, a)$, while the answer sets of $\Pi(\mathcal{SD}, \sigma_0, a)$ provide the semantics of $\mathcal{SD}$ in terms of the transition $< \sigma_0, a, \sigma_1 >$.

- We first define the encoding $\Pi(\mathcal{SD})$ of system description $\mathcal{SD}$ which consists of the encoding of the signature of $\mathcal{SD}$ and rules obtained from statements of $\mathcal{SD}$.

Translating $\mathcal{AL}$ into ASP - The General Idea

- We then define the encoding $\Pi(\sigma_0, 0)$ and $\Pi(a, 0)$, where $\Pi(\sigma_0, 0)$ represents the initial state σ_0 in a logic program form, and $\Pi(a, 0)$ encodes the fact that the set of actions a is executable in σ_0.
- Finally, we define $\Pi(\mathcal{SD}, \sigma_0, a) = \Pi(\mathcal{SD}) \cup \Pi(\sigma_0, 0) \cup \Pi(a, 0)$.

Translating $\mathcal{AL}$ into ASP - Encoding$\mathcal{SD}$ Signature

The signature of a given $\mathcal{SD}$ contains: *statics*, *inertial fluents*, *defined fluents*, and *actions*.

$\mathcal{SD}$	$Signature(\mathcal{SD})$
static g	g
inertial fluent f	$fluent(inertial, f)$
defined fluent f	$fluent(defined, f)$
action a_i	$action(a_i)$

Translating $\mathcal{AL}$ into ASP - Encoding $\mathcal{SD}$ Statements

Now the encoding of statements of $\mathcal{SD}$ is defined as follows:

$\mathcal{SD}$ **Statements**	$\Pi(\mathcal{SD})$ **Rules**
	Setting step: $\#const\ \ n = 1.$ $step(0..n).$
For each causal law a **causes** l **if** $p_0, \cdots, p_m$	$holds(l, I+1) \leftarrow holds(p_0, I), \cdots,$ $holds(p_m, I),$ $occurs(a, I).$
For each state constraint l **if** $p_0, \cdots, p_m$	$holds(l, I) \leftarrow holds(p_0, I), \cdots,$ $holds(p_m, I).$
	CWA for defined fluents $*$ $-holds(F, I) \leftarrow fluent(defined, F),$ $not\ \ holds(F, I),$ $step(I).$

Translating $\mathcal{AL}$ into ASP - Encoding $\mathcal{SD}$ Statements

Continued.

$\mathcal{SD}$ **Statements**	$\Pi(\mathcal{SD})$ **Rules**
For each executability cond. **impossible** $a_0, \cdots, a_k$ **if** $p_0, \cdots, p_m$	$-occurs(a_0, I)$ *or* $\cdots$ *or* $-occurs(a_k, I) \leftarrow holds(p_0, I), \cdots, holds(p_m, I),$
	Inertia rules $*$ $holds(F, I+1) \leftarrow fluent(inertial, F), holds(F, I), not\ -holds(F, I+1), I < n.$ $-holds(F, I+1) \leftarrow fluent(inertial, F), -holds(F, I), not\ holds(F, I+1), I < n.$

Translating $\mathcal{AL}$ into ASP - Encoding $\mathcal{SD}$ Statements

Continued.

$\mathcal{SD}$ **Statements**	$\Pi(\mathcal{SD})$ **Rules**
For each $action(a)$	CWA for actions $*$ $-occurs(a, I) \leftarrow not\ occurs(a, I),$ $step(I), action(a).$

Translating $\mathcal{AL}$ into ASP - Encoding $\mathcal{SD}$ Initial State and Actions

$\mathcal{SD}$ **Initial State**	$\Pi(\sigma_0, 0)$ **Rules**
For each $l \in \sigma_0$	$holds(l, 0).$

$\mathcal{SD}$ **Actions**	$\Pi(\alpha, 0)$ **Rules**
For each $a_i \in a$	$occurs(a_i, 0).$

$$\Pi(\mathcal{SD}, \sigma_0, a) = \Pi(\mathcal{SD}) \cup \Pi(\sigma_0, 0) \cup \Pi(a, 0).$$

Translating $\mathcal{AL}$ into ASP - Encoding $\mathcal{SD}$ Initial State and Actions

Definition

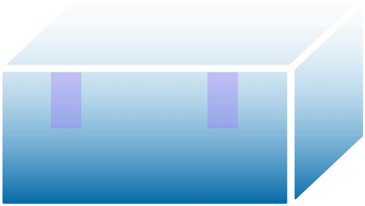
Let $\mathcal{SD}$ be a system description of $\mathcal{AL}$, σ_0 the initial state, and a a set of actions. $\Pi(\mathcal{SD}, \sigma_0, a)$ is specified as above. Then for transition $< \sigma_0, a, \sigma_1 >$, the *resulting state* σ_1 from this transition is defined as

$$\sigma_1 = \{l \mid holds(l, 1) \in A\},$$

where A is an answer set of $\Pi(\mathcal{SD}, \sigma_0, a)$.

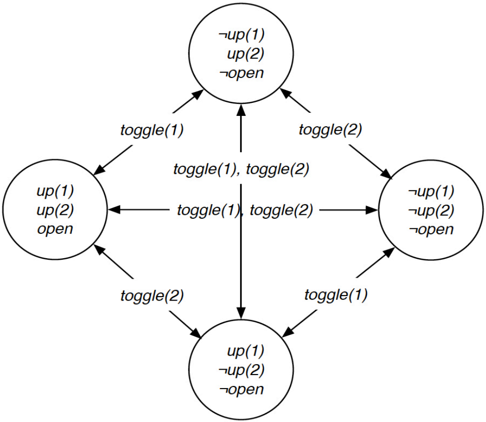
The Briefcase Example

The scenario: *Consider a briefcase with two clasps. We have an action, toggle, which moves a given clasp into the up position if the clasp is down, and vice versa. If both clasps are in the up position, the briefcase is open; otherwise, it is closed.*



The Briefcase Example

Briefcase domain: the transition diagram



The Briefcase Example

Briefcase domain: Signature

- sort *clasp* $= \{1, 2\}$
- inertial fluent $up(C)$ which holds if clasp C is up
- defined fluent *open* which holds if both clasps are up
- action $toggle(C)$ which toggles clasp C
- Note C can be 1 and 2 in above

The Briefcase Example

Briefcase domain: System Description

$\mathcal{SD}_{bc}$ contains the following statements:

Causal laws:
$toggle(1)$ **causes** $up(1)$ **if** $\neg up(1)$
$toggle(2)$ **causes** $up(2)$ **if** $\neg up(2)$
$toggle(1)$ **causes** $\neg up(1)$ **if** $up(1)$
$toggle(2)$ **causes** $\neg up(2)$ **if** $up(2)$
State Constraint:
$open$ **if** $up(1), up(2)$

The Briefcase Example

Briefcase domain: State

We assume the initial state $\sigma_0 = \{\neg up(1), up(2), \neg open\}$.

Briefcase domain: Transition

$< \sigma_0, \{toggle(1)\}, \sigma_1 >$

The Briefcase Example

- In order to obtain σ_1, we construct an ASP program $\Pi(\mathcal{SD}_{bc}, \sigma_0, \{toggle(1)\})$, and then compute its answer sets
- Recall that

$$\Pi(\mathcal{SD}_{bc}, \sigma_0, \{toggle(1)\}) =$$
$$\Pi(\mathcal{SD}_{bc}) \cup \Pi(\sigma_0, 0) \cup \Pi(\{toggle(1)\}, 0)$$

The Briefcase Example

Encoding $\mathcal{SD}_{bc}$: $\Pi(\mathcal{SD}_{bc})$

```
%% Signature
fluent(inertial,up(1)).
fluent(inertial,up(2)).
fluent(defined,open).

%% Setting step and clasp numbers.
#const n=1.
step(0..n).
#const c=2.
clasp(1..c).
```

The Briefcase Example

Encoding $\mathcal{SD}_{bc}$: $\Pi(\mathcal{SD}_{bc})$

```
%% Causal laws
%% %% toggle(C) causes up(C) if -up(C)
holds(up(C), I+1) :- occurs(toggle(C),I),
                     -holds(up(C),I).


%% toggle(C) causes -up(C) if up(C)
-holds(up(C), I+1) :- occurs(toggle(C),I),
                      holds(up(C),I).

%% State constraint
holds(open,I) :- holds(up(1),I), holds(up(2),I).
```

The Briefcase Example

```
%% CWA for defined fluents
-holds(F,I) :- fluent(defined,F),
               not holds(F,I), step(I).

%% Inertial rules
holds(F,I+1) :- fluent(inertial,F),
                holds(F,I),
                not -holds(F,I+1),
                I < n.

-holds(F,I+1) :- fluent(inertial,F),
                 -holds(F,I),
                 not holds(F,I+1),
                 I < n.
```

The Briefcase Example

```
%% CWA for actions
-occurs(toggle(C),I) :- not occurs(toggle(C),I),
                        step(I), clasp(C).
```

Encoding σ_0: $\Pi(\sigma_0, 0)$:

```
%% Initial state
-holds(up(1),0).
holds(up(2),0).
-holds(open,0).
```

The Briefcase Example

Encoding σ_0: $\Pi(\sigma_0, 0)$:

```
%% Initial state
-holds(up(1),0).
holds(up(2),0).
-holds(open,0).
```

Encoding actions: $\Pi(\{\mathit{toggle}(1)\}, 0)$:

```
%% Actions
occurs(toggle(1),0).
```

The Briefcase Example

Very importantly, in *clingo*, we need to make every rule be *safe*, i.e., each variable occurring in the head, must also occur in the positive body.

The Briefcase Example

Note: If we allow two toggle actions to be executed at the same time, the briefcase will never be opened from initial state:

$$\sigma_0 = \{\neg up(1), up(2), \neg open\}!$$

We name this program *briefcase.lp*. Running this program under *clingo*.

Consider the Blocks World we studied earlier. Suppose the Blocks World has the following system description $\mathcal{SD}_{BW}$:

Causal law:

$put(B, L)$ **causes** $on(B, L)$

State constraints:

$\neg on(B, L_2)$ **if** $on(B, L_1), L_1 \neq L_2$

$\neg on(B_2, B)$ **if** $on(B_1, B), B_1 \neq B_2$

$above(B, L)$ **if** $on(B, L)$

$above(B, L)$ **if** $on(B, B_1), on(B_1, L)$

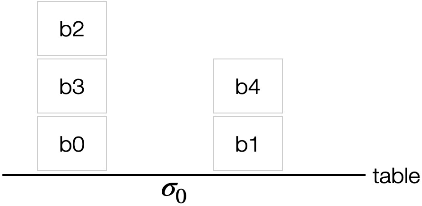
Executability conditions:

impossible $put(B, L)$ **if** $on(B_1, B)$

impossible $put(B_1, B)$ **if** $on(B_2, B)$

Tutorial and Lab Exercises

1. Encode $\mathcal{SD}_{BW}$ to $\Pi(\mathcal{SD}_{BW})$.
2. Suppose σ_0 is shown as follows:



Encode σ_0 and action $put(b_2, b_4)$ to $\Pi(\sigma_0, 0)$ and $\Pi(\{put(b_2, b_4)\}, 0)$, respectively. Then run program $\Pi(\mathcal{SD}_{BW}, \sigma_0, \{put(b_2, b_4)\})$ on *clingo*.