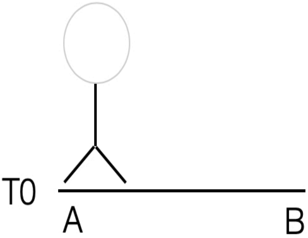
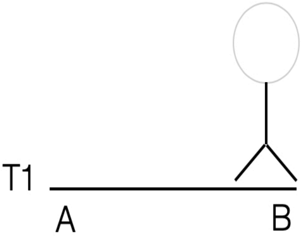
Lecture Eight: Modeling Dynamic Domains (I)

301315 Knowledge Representation and Reasoning

What is a Dynamic Domain - Concept and Definition

- A *domain* is a collection of (different types of) elements.
- The *state* of a domain is a snapshot about this domain at particular time.
- A domain is *dynamic* if the state of the domain is changing with time.





What is a Dynamic Domain - Concept and Definition

- We want our agents to hold their own in a changing environment.
- To do this, they need to be able to predict the effects of complex actions.
- And for that, they need to know about actions and their effects.

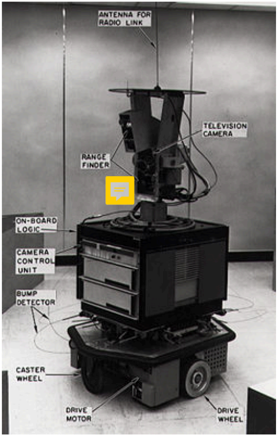
Question: How do we model such dynamic domain from an intelligent agent viewpoint?

What is a Dynamic Domain - Overview

- Let's start by looking at a simple example of an agent trying to act in a changing environment.
- The plan is to flush out some of the issues that exist in designing such an agent.
- Next, we'll look at a formal theory of actions and change.
- The theory views the world as a dynamic system whose states are changed by actions.
- Language $\mathcal{AL}$ describes these systems.
- $\mathcal{AL}$ has a direct translation into ASP.

Shakey - Early Research

- This is Shakey the robot built by SRI International (currently the Stanford Research Institute). It builds with blocks.



Shakey - The Early Research

- Building Shakey, and getting it to plan its actions based on a goal given it by humans brought to light many challenges involved in building and programming such a machine.

Shakey - The Early Research

- Hardware challenges:
  - creating a robotic arm
  - creating a robotic eye
  - We now know that much of those challenges are software challenges too.
- Software challenges:
  - How do we represent knowledge?
  - How do we teach a robot to use this knowledge to make plans?
  - How do we teach a robot to evaluate if its execution of a plan was successful?
  - How should the robot re-plan if there are changes?

Shakey - The Early Research

- Shakey only reasoned about blocks!
- Shakey used LISP and a theorem-proving planner called STRIPS.
- Its planning was domain-specific.
- This was the case for many planners that followed.
- We have learned a lot since then (thanks, in part, to Shakey).

Here is a recent Atlas robot stacking boxes.
`https://www.youtube.com/watch?v=rVlhMGQgDkY`

The Blocks World in ASP - Defining the Problem

We create a domain specific blocks world representation in ASP and to demonstrate how we can use ASP to model dynamic domains in general.

The Blocks World in ASP - Defining the Problem

While we do this, let us

- consider the types of knowledge we need to represent,
- develop a vocabulary for talking about dynamic domains,
- get motivated for the rigorous formalism.

The Blocks World in ASP - Defining the Problem

The Domain setting:

- We have a robotic arm that can manipulate configurations of same-sized cubic blocks on a table.
- It can move *unoccupied blocks*, one at a time, onto other unoccupied blocks or onto the table.
- At any given step, a block can be in at most one location.

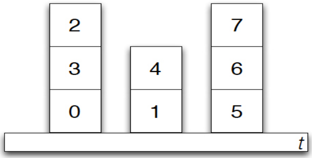
The Blocks World in ASP - Defining the Problem

- Towers can be as big as we want.
- The table is large enough to fit all blocks, even if they are not stacked.
- We do not take into account spacial relationships of towers, just which *blocks* are on top of each other and which blocks are on the table.

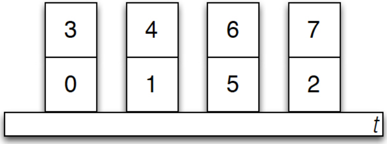
The Blocks World in ASP - Agent Specifications

Example

Given information that describes the following:



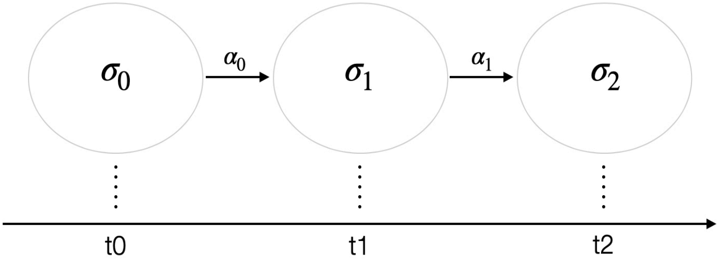
and actions that occur (put 2 on the table, put 7 on 2), know that this is what the configuration looks like now:



The Blocks World in ASP - Agent Specifications

Considerations:

- Objects: What are the objects that the agent will be reasoning about?
- Relations (predicates): What are the relations between them?
- Changes: How to represent the changes ?



Changing configurations

We can think of the initial configuration as a collection of terms that describe positions of blocks:

$$\begin{aligned}\sigma_0 = \quad & \{on(b_0, t), on(b_3, b_0), on(b_2, b_3),\\ & on(b_1, t), on(b_4, b_1), on(b_5, t),\\ & on(b_6, b_5), on(b_7, b_6)\}.\end{aligned}$$

Then, after action $put(b_2, t)$ is performed, the configuration changes to

$$\begin{aligned}\sigma_1 = \quad & \{on(b_0, t), on(b_3, b_0), on(b_1, t),\\ & on(b_4, b_1), on(b_2, t), on(b_5, t),\\ & on(b_6, b_5), on(b_7, b_6)\}.\end{aligned}$$

By performing action $put(b_7, b_2)$ on σ_1, we get the next configuration:

$$\begin{aligned}\sigma_2 = \quad & \{on(b_0, t), on(b_3, b_0), on(b_1, t),\\ & on(b_4, b_1), on(b_2, t), on(b_5, t),\\ & on(b_6, b_5), \color{red}{on(b_7, b_2)}\}.\end{aligned}$$

The Blocks World in ASP - Trajectory

The execution of a sequence of actions of the type $put(B, L)$ in configuration σ_0 determines the system's *trajectory*:

$$< \sigma_0, put(b_2, t), \sigma_1, put(b_7, b_2), \sigma_2 >$$

which describes its behaviour.

The Blocks World in ASP - Trajectory

- By *trajectory*, we mean a sequence

  $$< \sigma_0, \alpha_0, \sigma_1, \cdots, \sigma_{n-1}, \alpha_{n-1}, \sigma_n >,$$

  where $< \sigma_i, \alpha_i, \sigma_{i+1} >$ is a state-action-state transition of the system.

The Blocks World in ASP - Language for Describing the Systems

General considerations:

- Use integers from 0 to a finite n to denote the steps of the corresponding trajectories.
- Distinguish between
  - **fluents** - properties that can be changed by actions
  - **statics** - properties that never change
- In the blocks world, we chose to view the property of a block being on top of a location as a fluent and something being a block as a static.
- Each action takes 1 step to complete.

The Blocks World in ASP - Language for Describing the Systems

Relations:

- $hold(\#fluent, \#step)$ describes what fluents are true at a given step.
- $occur(\#action, \#step)$ describes what action occurred at a given step.
- In our example, we define:
  - $holds(on(B, L), I)$: block B is on location L at step I;
  - $occurs(put(B, L), I)$: block B was put on location L at step I.

The Blocks World in ASP - Language for Describing the Systems

Defining terms (objects) and predicates

```
#const n = 2.
step(0..n).

%% We have the following objects (they are not
%% in our clingo program):
%% #block = {b0,b1,...,b7}
%% #loc = {b0,b1,...,b7,t} %t means table.
%% #fluent = on(#block(X),#location(Y)):X!=Y.
%% #action = put(#block(X),#location(Y)):X!=Y.

%% We define the following predicates:
%% holds(#fluent,#step).
%% occurs(#action,#step).
```

The Blocks World in ASP - Language for Describing the Systems

```
block(b0). block(b1). block(b2). block(b3).
block(b4). block(b5). block(b6). block(b7).

loc(b0). loc(b1). loc(b2). loc(b3). loc(b4).
loc(b5). loc(b6). loc(b7). loc(t).
```

The Blocks World in ASP - Language for Describing the Systems

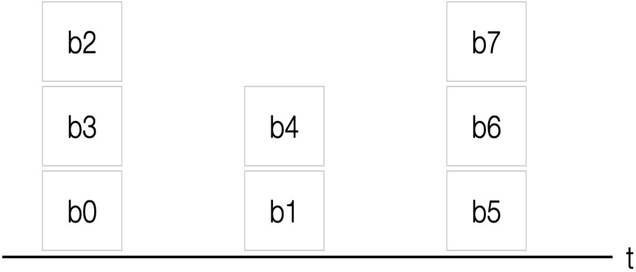
clingo program

PART I: Initial Configuration

```
%% holds(on(B,L),I): a block B is on location L
%% at step I.
holds(on(b0,t),0). holds(on(b3,b0),0).
holds(on(b2,b3),0). holds(on(b1,t),0).
holds(on(b4,b1),0). holds(on(b5,t),0).
holds(on(b6,b5),0). holds(on(b7,b6),0).

%% If block B is not known to be on
%% location L at step 0, then we
%% assume it is not.
-holds(on(B,L),0) :- not holds(on(B,L),0).
```

The Blocks World in ASP - Language for Describing the Systems

The initial configuration is illustrated as follows:



The Blocks World in ASP - Defining Blocks World Laws

PART II: Defining BW laws:

```
%% Putting block B on location L at step I
%% causes B to be on L at step I + 1.
holds(on(B,L),I+1) :- occurs(put(B,L),I).

%% A block cannot be in two locations at once
-holds(on(B,L2),I) :- holds(on(B,L1),I),
                      L1 != L2.

%% Only one block can be directly on top of another.
-holds(on(B2,B),I) :- block(B), % not the table
                      holds(on(B1,B),I),
                      B1 != B2.
```

The Blocks World in ASP - Defining Blocks World Laws

Now we add the following action statement into PART I + PART II

```
occurs(put(b2,t),0).
```

to form program Π^0_{BW}. What would we expect to know from the program?

- Query `holds(on(b2,t),1)`, returns yes, i.e., $\Pi^0_{BW} \models holds(on(b2,t),1)$. So far so good.
- But for query `holds(on(b0,t),1)`, returns unknown, i.e., $\Pi^0_{BW} \not\models holds(on(b0,t),1)$ even though we expect it to still be true.
- Why do we expect it to be true?

10 min classroom exercise

Let's discuss how we can derive query answers by computing answer set(s) of program Π^0_{BW}.

The Blocks World in ASP - Defining Blocks World Laws

A commonsense: *Normally* things are staying as they are, unless they are explicitly changed. *Inertia rules* capture this intuition.

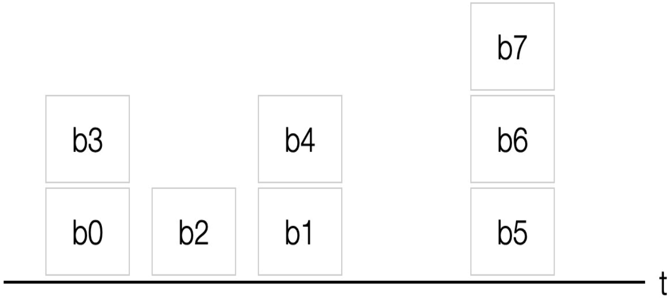
```
%% Inertia rule 1: anything that holds at step I,
%% will also hold at step I+1, as long as no evidence
%% shows its opposite
holds(F,I+1) :- holds(F,I),
                not -holds(F,I+1), I<n.

%% Inertia rule 2: anything that does not hold at
%% step I, will also not hold at step I+1, as long
%% as no evidence shows its opposite
-holds(F,I+1) :- -holds(F,I),
                 not holds(F,I+1), I<n.
```

The Blocks World in ASP - Defining Blocks World Laws

Note that inertia rules are a typical default representation we studied earlier. Now by adding these two inertia rules into the program, we call this newly formed program Π^1_{BW}, then

- query `holds(on(b0,t),1)`, returns yes, i.e., $\Pi^1_{BW} \models holds(on(b0,t),1)$.

The configuration is changed, as illustrated in the following:



The Blocks World in ASP - Defining More Blocks World Laws

Defining impossible actions

- Quite often, we need to express that some actions are not possibly executed in the domain. In our example, action `occurs(put(b6,t),0)` should not be allowed, because of `holds(on(b7,b6),0)` - block b6 is occupied.
- But the current system (program) does not prevent these impossible actions to happen.
- We need more rules to express this!

The Blocks World in ASP - Defining More Blocks World Laws

Impossible actions:

```
%% It is impossible to move an occupied block:
-occurs(put(B,t),I) :- holds(on(B1,B),I).
-occurs(put(B,B2),I) :- holds(on(B1,B),I),
                        holds(on(B2,B3),I),
                        B1 != B2, B != B3.
%% It is impossible to move a block onto
%% an occupied block:
-occurs(put(B1,B),I) :- holds(on(B1,L),I),
                        holds(on(B2,B),I),
                        B1 != B2, B != t.
```

- Important notice: we need to make each rule be *safe*.

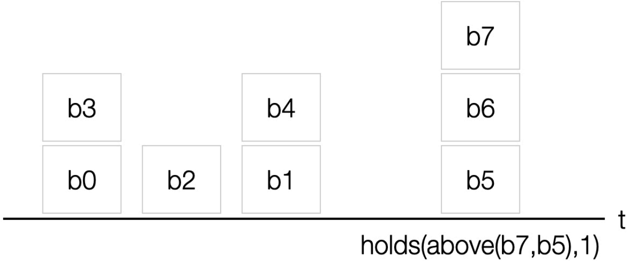
The Blocks World in ASP - Defining More Blocks World Laws

Now our Blocks World program computes what we want:

- The direct effects of actions `put(B,L)`.
- Indirect effects of actions `put(BL)`, derived from essential BW laws.
- The Inertia rules: normally things stay as they are.
- Which actions should not be allowed.

The Blocks World in ASP - Enhancing the Blocks World

- It is much convenient if we can express some block is *above* another. For instance, if we know facts `holds(on(b1,b0),0)` and `holds(on(b2,b1),0)`, we would like to say that block `b2` is *above* block `b0`.
- In fact, we can define the new fluent `above(#block,#location)` in terms of fluent `on(#block,#location)`.



Defining new fluent `above(#block,#location)`:

```
%% Block B is located above location L:
holds(above(B,L),I) :- holds(on(B,L),I).
holds(above(B,L),I) :- holds(on(B,B1),I),
                       holds(above(B1,L),I).
%% CWA for above:
-holds(above(B,L),I) :- not holds(above(B,L),I).
```

We add the above rules into Π^1_{BW} to form a new version BW program Π^2_{BW}.

Tutorial and Lab Exercises

1. Based on the Blocks World we have studied in this lecture, write a complete *clingo* program which contains all rules from Π^2_{BW} *plus* one more rule:

   ```
   occurs(put(b4,b7),1).
   ```

2. By running your *clingo* program, what is the answer for query `holds(above(b4,b6),s)`?

Tutorial and Lab Exercises

3. Given the following story: Bob has requirements for playing with his iPad: he should make sure that his homework is done, the bed is made, and he has practiced Tae Kwon Do. He can only do one thing at a time. Of course, he cannot make the bed if it already made or do his homework if it is already done or if none was assigned.
   (a) Select an initial situation and a sequence of Bob's actions which would allow him to play the iPad.
   (b) You can define actions as terms `do(hw)`, `do(make_bed)`, *do(tkd)* and `do(iPad)` to make it easy to express the requirements. Also note that Bob can only do one action at a time.
   (c) Write a *clingo* program to represent the above dynamic domain.