

Chapter 14:

Planning with relational actions

Relational Actions and Planning

- Agents reason in time.
- Agents reason about time.
- Time passes as an agent acts and reasons.
- Given a goal, it is useful for an agent to think about what it will do in the future to determine what it will do now.
- Relational representations allow to reason about actions even before the individual objects of the domain become known to the agent.

Representing Time

Time can be modeled in a number of ways:

Discrete time Time is modeled as jumping from one time point to another.

Continuous time Time is modeled as being dense.

Event-based time Time steps don't have to be uniform; time steps can be between interesting events.

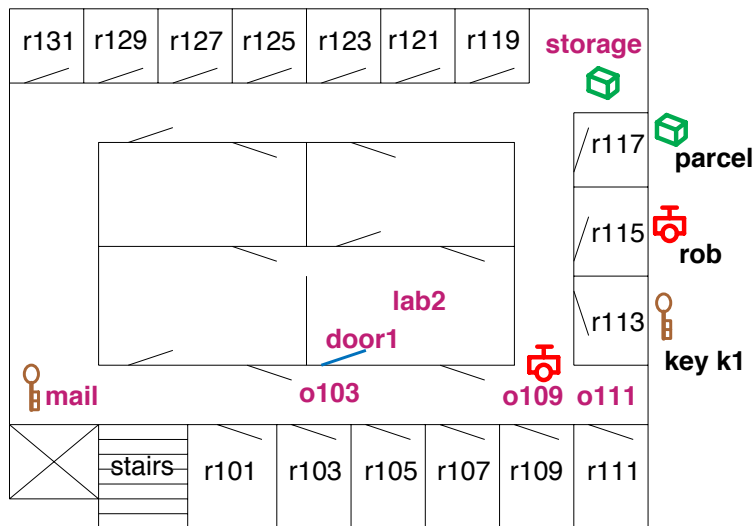
State space Instead of considering time explicitly, actions can map from one state to another.

You can model time in terms of **points** or **intervals**.

When modeling relations, you distinguish two basic types:

- **Static relations** are those relations whose value does not depend on time.
- **Dynamic relations** are relations whose truth values depends on time. Either
 - ▶ **derived relations** whose definition can be derived from other relations for each time,
 - ▶ **primitive relations** whose truth value can be determined by considering previous times.

The Delivery Robot World



Modeling the Delivery Robot World

- **Individuals:** rooms, doors, keys, parcels, and the robot.
- **Actions:**
 - move from room to room
 - pick up and put down keys and packages
 - unlock doors (with the appropriate keys)
- **Relations:** represent
 - the robot's position
 - the position of packages and keys and locked doors
 - what the robot is holding

Example Relations

- $at(Obj, Loc)$ is true in a situation if object Obj is at location Loc in the situation.
- $carrying(Ag, Obj)$ is true in a situation if agent Ag is carrying Obj in that situation.
- $sitting_at(Obj, Loc)$ is true in a situation if object Obj is sitting on the ground (not being carried) at location Loc in the situation.
- $unlocked(Door)$ is true in a situation if door $Door$ is unlocked in the situation.
- $autonomous(Ag)$ is true if agent Ag can move autonomously. This is static.

Example Relations (cont.)

- $opens(Key, Door)$ is true if key Key opens door $Door$. This is static.
- $adjacent(Pos_1, Pos_2)$ is true if position Pos_1 is adjacent to position Pos_2 so that the robot can move from Pos_1 to Pos_2 in one step.
- $between(Door, Pos_1, Pos_2)$ is true if $Door$ is between position Pos_1 and position Pos_2 . If the door is unlocked, the two positions are adjacent.

- $move(Ag, From, To)$ agent Ag moves from location $From$ to adjacent location To . The agent must be sitting at location $From$.
- $pickup(Ag, Obj)$ agent Ag picks up Obj . The agent must be at the location that Obj is sitting.
- $putdown(Ag, Obj)$ the agent Ag puts down Obj . It must be holding Obj .
- $unlock(Ag, Door)$ agent Ag unlocks $Door$. It must be outside the door and carrying the key to the door.

sitting_at(rob, o109).

sitting_at(parcel, storage).

sitting_at(k1, mail).

between(door1, o103, lab2).

opens(k1, door1).

autonomous(rob).

$at(Obj, Pos) \leftarrow sitting_at(Obj, Pos).$

$at(Obj, Pos) \leftarrow carrying(Ag, Obj) \wedge at(Ag, Pos).$

$adjacent(o109, o103).$

$adjacent(o103, o109).$

...

$adjacent(lab2, o109).$

$adjacent(P_1, P_2) \leftarrow$
 $between(Door, P_1, P_2) \wedge$
 $unlocked(Door).$

Function Symbols

- Often we want to refer to individuals in terms of components.
- Examples: 4:55 p.m. English sentences. A classlist.
- We extend the notion of **term**. So that a term can be $f(t_1, \dots, t_n)$ where f is a **function symbol** and the t_i are terms.
- In an interpretation and with a variable assignment, term $f(t_1, \dots, t_n)$ denotes an individual in the domain.
- Function symbols can be recursively embedded:
 - ▶ One function symbol and one constant can refer to infinitely many individuals.
 - ▶ Function symbols can be used to describe sequential structures.

Example: Lists

- A list is an ordered sequence of elements.
- Let's use the constant `nil` to denote the empty list, and the function `cons(H, T)` to denote the list with first element H and rest-of-list T . **These are not built-in.**
- The list containing *sue*, *kim* and *randy* is

`cons(sue, cons(kim, cons(randy, nil)))`

- `append(X, Y, Z)` is true if list Z contains the elements of X followed by the elements of Y

`append(nil, Z, Z).`

`append(cons(A, X), Y, cons(A, Z)) ← append(X, Y, Z).`

- State-based representation where the states are denoted by terms.
- A **situation** is a term that denotes a state.
- There are two ways to refer to states:
 - ▶ **init** denotes the initial state
 - ▶ **do(A, S)** denotes the state resulting from doing action A in state S , if it is possible to do A in S .
- Time is reified by means of situations which can be entered.
 - ▶ Individuals represent points in time.
- A situation encodes how to get to the state it denotes.
 - ▶ A state may be represented by multiple situations.
 - ▶ A state may be represented by no situations if it is unreachable.
 - ▶ A situation may represent no states, if an action was not possible.

Example Situations

- *init*
- *do(move(rob, o109, o103), init)*
- *do(move(rob, o103, mail),
do(move(rob, o109, o103),
init)).*
- *do(pickup(rob, k1),
do(move(rob, o103, mail),
do(move(rob, o109, o103),
init))).*

- Add an extra term to each dynamic predicate indicating the situation.
- Example Atoms:

at(rob, o109, init)

at(rob, o103, do(move(rob, o109, o103), init))

at(k1, mail, do(move(rob, o109, o103), init))

Axiomatizing using the Situation Calculus

- You specify what is true in the **initial state** using axioms with *init* as the situation parameter.
- **Primitive relations** are axiomatized by specifying what is true in situation $do(A, S)$ in terms of what holds in situation S .
- **Derived relations** are defined using clauses with a free variable in the situation argument.
- **Static relations** are defined without reference to the situation.

sitting_at(rob, o109, init).

sitting_at(parcel, storage, init).

sitting_at(k1, mail, init).

adjacent(P₁, P₂, S) ←
between(Door, P₁, P₂) ∧
unlocked(Door, S).

adjacent(lab2, o109, S).

...

When are actions possible?

$poss(A, S)$ is true if action A is possible in situation S .

$$poss(putdown(Ag, Obj), S) \leftarrow \\ carrying(Ag, Obj, S).$$
$$poss(move(Ag, Pos_1, Pos_2), S) \leftarrow \\ autonomous(Ag) \wedge \\ adjacent(Pos_1, Pos_2, S) \wedge \\ sitting_at(Ag, Pos_1, S).$$

Example: Unlocking the door makes the door unlocked:

$$\begin{aligned} \text{unlocked}(\text{Door}, \text{do}(\text{unlock}(\text{Ag}, \text{Door}), S)) \leftarrow \\ \text{poss}(\text{unlock}(\text{Ag}, \text{Door}), S). \end{aligned}$$

Frame Axiom: No actions lock the door:

$$\begin{aligned} \text{unlocked}(\text{Door}, \text{do}(A, S)) \leftarrow \\ \text{unlocked}(\text{Door}, S) \wedge \\ \text{poss}(A, S). \end{aligned}$$

Example: axiomatizing *carried*

Picking up an object causes it to be carried:

$$\begin{aligned} \text{carrying}(Ag, Obj, \text{do}(\text{pickup}(Ag, Obj), S)) \leftarrow \\ \text{poss}(\text{pickup}(Ag, Obj), S). \end{aligned}$$

Frame Axiom: The object is being carried if it was being carried before unless the action was to put down the object:

$$\begin{aligned} \text{carrying}(Ag, Obj, \text{do}(A, S)) \leftarrow \\ \text{carrying}(Ag, Obj, S) \wedge \\ \text{poss}(A, S) \wedge \\ A \neq \text{putdown}(Ag, Obj). \end{aligned}$$

Example: *sitting_at*

An object is sitting at a location if:

- it moved to that location:

$$\textit{sitting_at}(\textit{Obj}, \textit{Pos}, \textit{do}(\textit{move}(\textit{Obj}, \textit{Pos}_0, \textit{Pos}), \textit{S})) \leftarrow \\ \textit{poss}(\textit{move}(\textit{Obj}, \textit{Pos}_0, \textit{Pos})).$$

- it was put down at that location:

$$\textit{sitting_at}(\textit{Obj}, \textit{Pos}, \textit{do}(\textit{putdown}(\textit{Ag}, \textit{Obj}), \textit{S})) \leftarrow \\ \textit{poss}(\textit{putdown}(\textit{Ag}, \textit{Obj}), \textit{S}) \wedge \\ \textit{at}(\textit{Ag}, \textit{Pos}, \textit{S}).$$

- it was at that location before and didn't move and wasn't picked up.

More General Frame Axioms

The only actions that undo *sitting_at* for object *Obj* is when *Obj* moves somewhere or when someone is picking up *Obj*.

$$\begin{aligned} \textit{sitting_at}(\textit{Obj}, \textit{Pos}, \textit{do}(A, S)) \leftarrow \\ \textit{poss}(A, S) \wedge \\ \textit{sitting_at}(\textit{Obj}, \textit{Pos}, S) \wedge \\ \forall \textit{Pos}_1 \quad A \neq \textit{move}(\textit{Obj}, \textit{Pos}, \textit{Pos}_1) \wedge \\ \forall \textit{Ag} \quad A \neq \textit{pickup}(\textit{Ag}, \textit{Obj}). \end{aligned}$$

More General Frame Axioms

$$\forall Ag \ A \neq \text{pickup}(Ag, Obj)$$

is equivalent to:

$$\sim \exists Ag \ A = \text{pickup}(Ag, Obj)$$

which can be implemented as

$$\begin{aligned} \text{sitting_at}(Obj, Pos, do(A, S)) \leftarrow \\ \dots \wedge \dots \wedge \dots \wedge \\ \sim \text{is_pickup_action}(A, Obj). \end{aligned}$$

with the clause:

$$\begin{aligned} \text{is_pickup_action}(A, Obj) \leftarrow \\ A = \text{pickup}(Ag, Obj). \end{aligned}$$

which is equivalent to:

$$\text{is_pickup_action}(\text{pickup}(Ag, Obj), Obj).$$

Given

- an initial world description
- a description of available actions
- a goal

a **plan** is a sequence of actions that will achieve the goal.

Example Planning

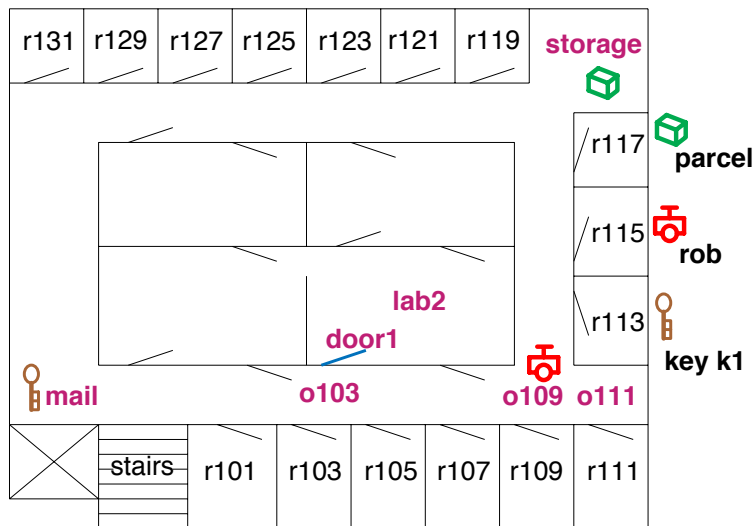
If you want a plan to achieve Rob holding the key $k1$ and being at $o103$, the query

$? \text{carrying}(\text{rob}, k1, S) \wedge \text{at}(\text{rob}, o103, S).$

has an answer

$$S = \text{do}(\text{move}(\text{rob}, \text{mail}, o103), \\ \text{do}(\text{pickup}(\text{rob}, k1), \\ \text{do}(\text{move}(\text{rob}, o103, \text{mail}), \\ \text{do}(\text{move}(\text{rob}, o109, o103), \text{init}))))).$$

The Delivery Robot World



- **Idea:** backward chain on the situation calculus rules.
- A complete search strategy (e.g., A^* or iterative deepening) is guaranteed to find a solution.
- When there is a solution to the query with situation $S = do(A, S_1)$, action A is the last action in the plan.
- You can virtually always use a frame axiom so that the search space is largely unconstrained by the goal. Search space is enormous.

- Given a goal, you would like to consider only those actions that actually achieve it.
- Example:

? $carrying(rob, parcel, S) \wedge in(rob, lab2, S)$.

the last action needed is irrelevant to the left subgoal.

- So we need to combine the planning algorithms with the relational representations.