Chapter 8: Actions and Planning

- Lecture 1 Actions, planning and the robot planning domain
- Lecture 2 The STRIPS representation
- Lecture 3 The situation calculus.
- Lecture 4 Planning, forward and resolution planning.
- Lecture 5 The STRIPS planner.
- Lecture 6 Regression planner.

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.

Representing Time

Time can be modeled in a number of ways:

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

Continuous time You can model time as being dense.

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

State space Instead of considering time explicitly, you can consider actions as mapping from one state to another.

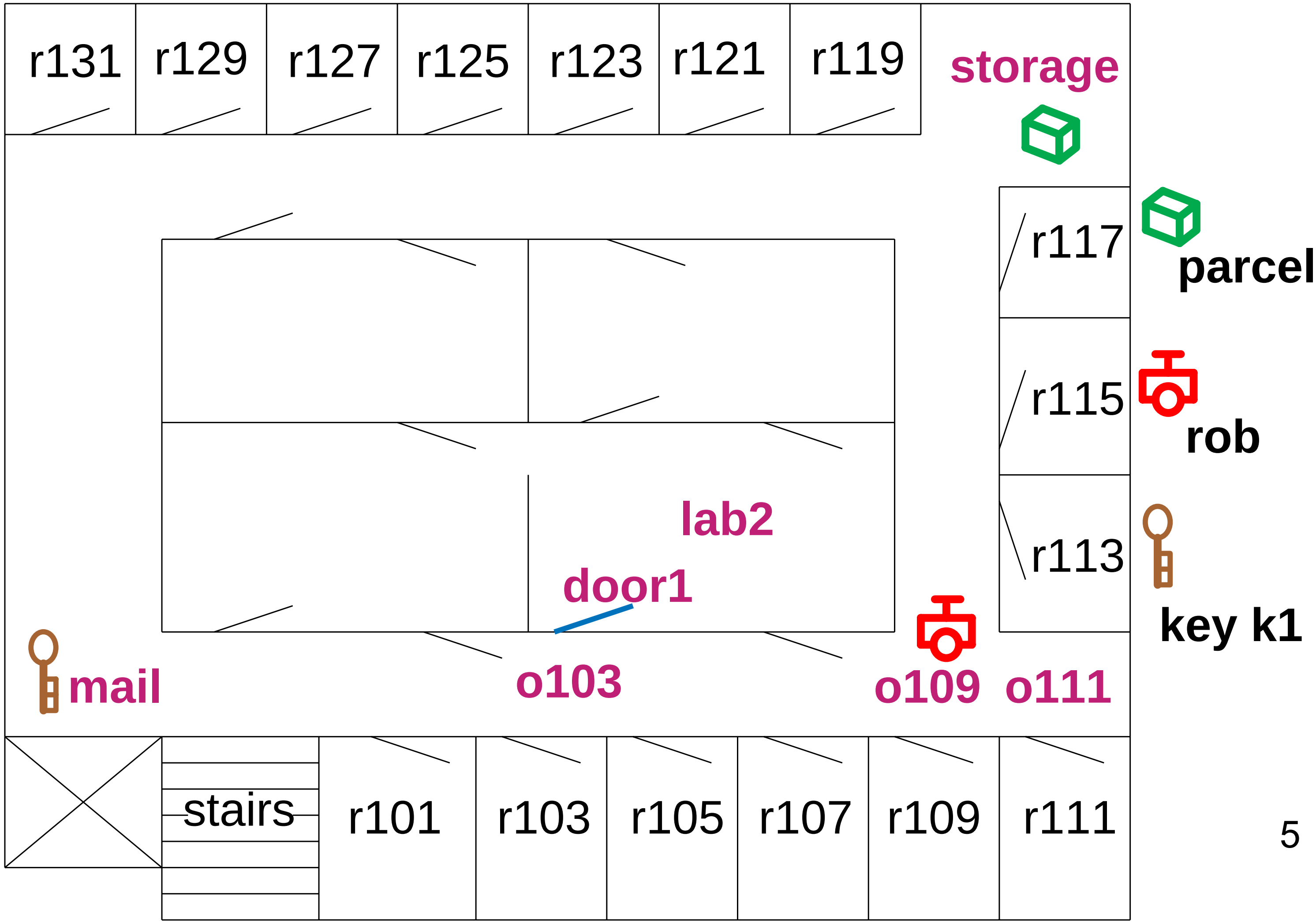
You can model time in terms of **points** or **intervals.**

Time and Relations

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.

- $opens(Key, Door)$ 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.

Actions

- *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.

Initial Situation

$sitting\_at(rob, o109).$

$sitting\_at(parcel, storage).$

$sitting\_at(k1, mail).$

Static Facts

$between(door1, o103, lab2).$

$opens(k1, door1).$

$autonomous(rob).$

Derived Relations

$$at(Obj, Pos) \leftarrow sitting\_at(Obj, Pos).$$

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

$$adjacent(o109, o103).$$

$$adjacent(o103, o109).$$

$$\ldots$$

$$adjacent(lab2, o109).$$

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

STRIPS Representation

- State-based view of time.
- The actions are external to the logic.
- Given a state and an action, the STRIPS representation is used to determine
  - whether the action can be carried out in the state
  - what is true in the resulting state

STRIPS Representation: Idea

- Predicates are primitive or derived.
- Use normal rules for derived predicates.
- The STRIPS representation is used to determine the truth values of primitive predicates based on the previous state and the action.
- Based on the idea that most predicates are unaffected by a single action.
- STRIPS assumption: Primitive relations not mentioned in the description of the action stay unchanged.

STRIPS Representation of an action

The STRIPS representation for an action consists of:

preconditions A list of atoms that need to be true for the action to occur

delete list A list of those primitive relations no longer true after the action

add list A list of the primitive relations made true by the action

STRIPS Representation of “pickup”

The action $pickup(Ag, Obj)$ can be defined by:

preconditions $[autonomous(Ag), Ag \neq Obj, at(Ag, Pos), sitting\_at(Obj, Pos)]$

delete list $[sitting\_at(Obj, Pos)]$

add list $[carrying(Ag, Obj)]$

STRIPS Representation of “move”

The action $move(Ag, Pos_1, Pos_2)$ can be defined by:

preconditions $[autonomous(Ag), adjacent(Pos_1, Pos_2, S),$
$sitting\_at(Ag, Pos_1)]$

delete list $[sitting\_at(Ag, Pos_1)]$

add list $[sitting\_at(Ag, Pos_2)]$

Example Transitions

$$\left\{\begin{array}{l} sitting\_at(rob, o109). \\ sitting\_at(parcel, storage). \\ sitting\_at(k1, mail). \end{array}\right\}$$

$$\xrightarrow{move(rob,\, o109,\, storage)} \left\{\begin{array}{l} sitting\_at(rob, storage). \\ sitting\_at(parcel, storage). \\ sitting\_at(k1, mail). \end{array}\right\}$$

$$\xrightarrow{pickup(rob,\, parcel)} \left\{\begin{array}{l} sitting\_at(rob, storage). \\ carrying(rob, parcel). \\ sitting\_at(k1, mail). \end{array}\right\}$$

Situation Calculus

- State-based representation where the states are denoted by terms.
- A situation is a term that dentotes 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.
- A situation also encodes how to get to the state it denotes.

Example States

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

Using the Situation Terms

- 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.

Initial Situation

$sitting\_at(rob, o109, init).$

$sitting\_at(parcel, storage, init).$

$sitting\_at(k1, mail, init).$

Derived Relations

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

$adjacent(lab2, o109, S).$

$\ldots$

When are actions possible?

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

$$poss(putdown(Ag, Obj), S) \leftarrow$$
$$carrying(Ag, Obj, S).$$

$$poss(move(Ag, Pos_1, Pos_2), S) \leftarrow$$
$$autonomous(Ag) \wedge$$
$$adjacent(Pos_1, Pos_2, S) \wedge$$
$$sitting\_at(Ag, Pos_1, S).$$

Axiomatizing Primitive Relations

Example: Unlocking the door makes the door unlocked:

$$unlocked(Door, do(unlock(Ag, Door), S)) \leftarrow$$
$$poss(unlock(Ag, Door), S).$$

Frame Axiom: No actions lock the door:

$$unlocked(Door, do(A, S)) \leftarrow$$
$$unlocked(Door, S) \wedge$$
$$poss(A, S).$$

Example: axiomatizing *carried*

Picking up an object causes it to be carried:

$$carrying(Ag, Obj, do(pickup(Ag, Obj), S)) \leftarrow$$
$$poss(pickup(Ag, Obj), S).$$

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

$$carrying(Ag, Obj, do(A, S)) \leftarrow$$
$$carrying(Ag, Obj, S) \wedge$$
$$poss(A, S) \wedge$$
$$A \neq putdown(Ag, Obj).$$

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*.

$$
\begin{aligned}
& sitting\_at(Obj, Pos, do(A, S)) \leftarrow \\
& \quad poss(A, S) \wedge \\
& \quad sitting\_at(Obj, Pos, S) \wedge \\
& \quad \forall Pos_1 \;\; A \neq move(Obj, Pos, Pos_1) \wedge \\
& \quad \forall Ag \;\; A \neq pickup(Ag, Obj).
\end{aligned}
$$

The last line is equivalent to:

$$\sim \exists Ag \; A = pickup(Ag, Obj)$$

which can be implemented as

$$sitting\_at(Obj, Pos, do(A, S)) \leftarrow$$
$$\cdots \wedge \cdots \wedge \cdots \wedge$$
$$\sim is\_pickup\_action(A, Obj).$$

with the clause:

$$is\_pickup\_action(A, Obj) \leftarrow$$
$$A = pickup(Ag, Obj).$$

which is equivalent to:

$$is\_pickup\_action(pickup(Ag, Obj), Obj).$$

STRIPS and the Situation Calculus

- Anything that can be stated in STRIPS can be stated in the situation calculus.
- The situation calculus is more powerful. For example, the "drop everything" action.
- To axiomatize STRIPS in the situation calculus, we can use $holds(C, S)$ to mean that C is true in situation S.

$holds(C, do(A, W)) \leftarrow$	
$\quad preconditions(A, P) \wedge$	The preconditions of
$\quad holdsall(P, W) \wedge$	of A all hold in W.
$\quad add\_list(A, AL) \wedge$	C is on the
$\quad member(C, AL).$	addlist of A.
$holds(C, do(A, W)) \leftarrow$	
$\quad preconditions(A, P) \wedge$	The preconditions of
$\quad holdsall(P, W) \wedge$	of A all hold in W.
$\quad delete\_list(A, DL) \wedge$	C isn't on the
$\quad notin(C, DL) \wedge$	deletelist of A.
$\quad holds(C, W).$	C held before A.

Planning

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 $o103$, you can issue the query

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

This has an answer

$$\begin{aligned} S = do(&move(rob, mail, o103), \\ &do(pickup(rob, k1), \\ &\quad do(move(rob, o103, mail), \\ &\quad\quad do(move(rob, o109, o103), init)))). \end{aligned}$$

Forward Planner

- Search in the state-space graph, where the nodes represent states and the arcs represent actions.
- Search from initial state to a state that satisfies the goal.
- A complete search strategy (e.g., A^* or iterative deepening) is guaranteed to find a solution.
- Branching factor is the number of legal actions. Path length is the number of actions to achieve the goal.
- You usually can't do backward planning in the state space, as the goal doesn't uniquely specify a state.

Planning as Resolution

- Idea: backward chain on the situation calculus rules or the situation calculus axiomatization of STRIPS.

- 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.

Goal-directed searching

- 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.

STRIPS Planner

- Divide and conquer: to create a plan to achieve a conjunction of goals, create a plan to achieve one goal, and then create a plan to achieve the rest of the goals.
- To achieve a list of goals:
  - choose one of them to achieve.
  - If it is not already achieved
    - choose an action that makes the goal true
    - achieve the preconditions of the action
    - carry out the action
  - achieve the rest of the goals.

STRIPS Planner Code

% $achieve\_all(Gs, W_1, W_2)$ is true if W_2 is the world resulting
% from achieving every element of the list Gs of goals from
% the world W_1.

$$achieve\_all([\,], W_0, W_0).$$

$$\begin{aligned}
&achieve_all(Goals, W_0, W_2) \leftarrow \\
&\quad remove(G, Goals, Rem_Gs) \wedge \\
&\quad achieve(G, W_0, W_1) \wedge \\
&\quad achieve_all(Rem_Gs, W_1, W_2).
\end{aligned}$$

% $achieve(G, W_0, W_1)$ is true if W_1 is the resulting world

% after achieving goal G from the world W_0.

$$
\begin{aligned}
& achieve(G, W, W) \leftarrow \\
& \qquad holds(G, W). \\
& achieve(G, W_0, W_1) \leftarrow \\
& \qquad clause(G, B) \wedge \\
& \qquad achieve\_all(B, W_0, W_1). \\
& achieve(G, W_0, do(Action, W_1)) \leftarrow \\
& \qquad achieves(Action, G) \wedge \\
& \qquad preconditions(Action, Pre) \wedge \\
& \qquad achieve\_all(Pre, W_0, W_1).
\end{aligned}
$$

Example of STRIPS-planning (1)

Query:

?achieve_all([carrying(rob, parcel), sitting_at(rob, lab2)], init, S)

Sequence of actions transforming initial state into goal state:

do(move(rob, o103, lab2),
do(unlock(rob, door1),
do(move(rob, mail, o103),
do(pickup(rob, k1, mail),
do(move(rob, o103, mail),
do(move(rob, o109, o103),
do(move(rob, storage, o109),

second goal

do(pickup(rob, parcel, storage),
do(move(rob, o109, storage),
init)))))))))).

first goal

Example of STRIPS-planning (2)

Query (subgoals in reversed order):

?achieve_all([sitting_at(rob, lab2), carrying(rob, parcel)], init, S)

Sequence of actions transforming initial state into goal state:

```
do(pickup(rob, parcel, storage),
    do(move(rob, o109, storage),
        do(move(rob, o103, o109),
            do(move(rob, lab2, o103),
                do(move(rob, o103, lab2),
                    do(unlock(rob, door1),
                        do(move(rob, mail, o103),
                            do(pickup(rob, k1, mail),
                                do(move(rob, o103, mail),
                                    do(move(rob, o109, o103),
                                        init)))))))))).
```

second goal undoes first goal!

first goal

Undoing Achieved Goals

Example: consider trying to achieve

$$[carrying(rob, parcel), sitting\_at(rob, lab2)]$$

Example: consider trying to achieve

$$[sitting\_at(rob, lab2), carrying(rob, parcel)]$$

- The STRIPS algorithm, as presented, is unsound.
- Achieving one subgoal may undo already achieved subgoals.

Fixing the STRIPS Algorithm

Two ideas to make STRIPS sound:

- **Protect subgoals** so that, once achieved, until they are needed, they cannot be undone. Let *remove* return different choices.
- **Reachieve subgoals** that have been undone.
  - Protecting subgoals makes STRIPS incomplete.
  - Reachieving subgoals finds longer plans than necessary.

Does protecting always work?

- **Example** Suppose the robot can only carry one item at a time. Consider the goal:

$$sitting\_at(rob, lab2) \land carrying(rob, parcel)$$

- We cannot consider the subgoals in isolation!

Regression

- **Idea:** don’t solve one subgoal by itself, but keep track of all subgoals that must be achieved.
- Given a set of goals:
  - If they all hold in the initial state, return the empty plan
  - Otherwise, choose an action *A* that achieves one of the subgoals. This will be the last action in the plan.
  - Determine what must be true immediately before *A* so that all of the goals will be true immediately after. Recursively solve these new goals.

Regression as Path Finding

- The nodes are sets of goals. Arcs correspond to actions.
- A node labeled with goal set G has a neighbor for each action A that achieves one of the goals in G.
- The neighbor corresponding to action A is the node with the goals G_A that must be true immediately before the action A so that all of the goals in G are true immediately after A. G_A is the weakest precondition for action A and goal set G.
- Search can stop when you have a node where all the goals are true in the initial state.

Weakest preconditions

$wp(A, GL, WP)$ is true if WP is the weakest precondition that must occur immediately before action A so every element of goal list GL is true immediately after A.

For the STRIPS representation (with all predicates primitive):

- $wp(A, GL, WP)$ is *false* if any element of GL is on delete list of action A.
- Otherwise WP is

$$preconds(A) \cup \{G \in GL : G \notin add\_list(A)\}.$$

where $preconds(A)$ is the list of preconditions of action A and $add\_list(A)$ is the add list of action A.

Weakest Precondition Example

The weakest precondition for

$$[sitting\_at(rob, lab2), carrying(rob, parcel)]$$

to be true after $move(rob, Pos, lab2)$ is that

$$\begin{aligned}[&autonomous(rob),\\ &\quad adjacent(Pos, lab2),\\ &\quad sitting\_at(rob, Pos),\\ &\quad carrying(rob, parcel)]\end{aligned}$$

is true immediately before the action.

A Regression Planner

% $solve(GL, W)$ is true if every element of goal list GL is true
% in world W.

$$
\begin{aligned}
& solve(GoalSet, init) \leftarrow \\
& \qquad holdsall(GoalSet, init). \\
& solve(GoalSet, do(Action, W)) \leftarrow \\
& \qquad consistent(GoalSet) \wedge \\
& \qquad choose\_goal(Goal, GoalSet) \wedge \\
& \qquad choose\_action(Action, Goal) \wedge \\
& \qquad wp(Action, GoalSet, NewGoalSet) \wedge \\
& \qquad solve(NewGoalSet, W).
\end{aligned}
$$

Regression Search Space Example

[*carrying*(*rob*,*parcel*), *sitting_at*(*rob*,*lab2*)]

pickup(*rob*,*parcel*)

move(*rob*,*P*,*lab2*)

[*sitting_at*(*parcel*,*lab2*), *sitting_at*(*rob*,*lab2*)]

[*carrying*(*rob*,*parcel*), *sitting_at*(*rob*,*P*), *adjacent*(*P*,*lab2*)]

=

[*carrying*(*rob*,*parcel*), *sitting_at*(*rob*,*o103*), *unlocked*(*door1*)]

unlock(*rob*,*door1*)

[*carrying*(*rob*,*parcel*), *sitting_at*(*rob*,*o103*), *carrying*(*rob*,*k1*)]