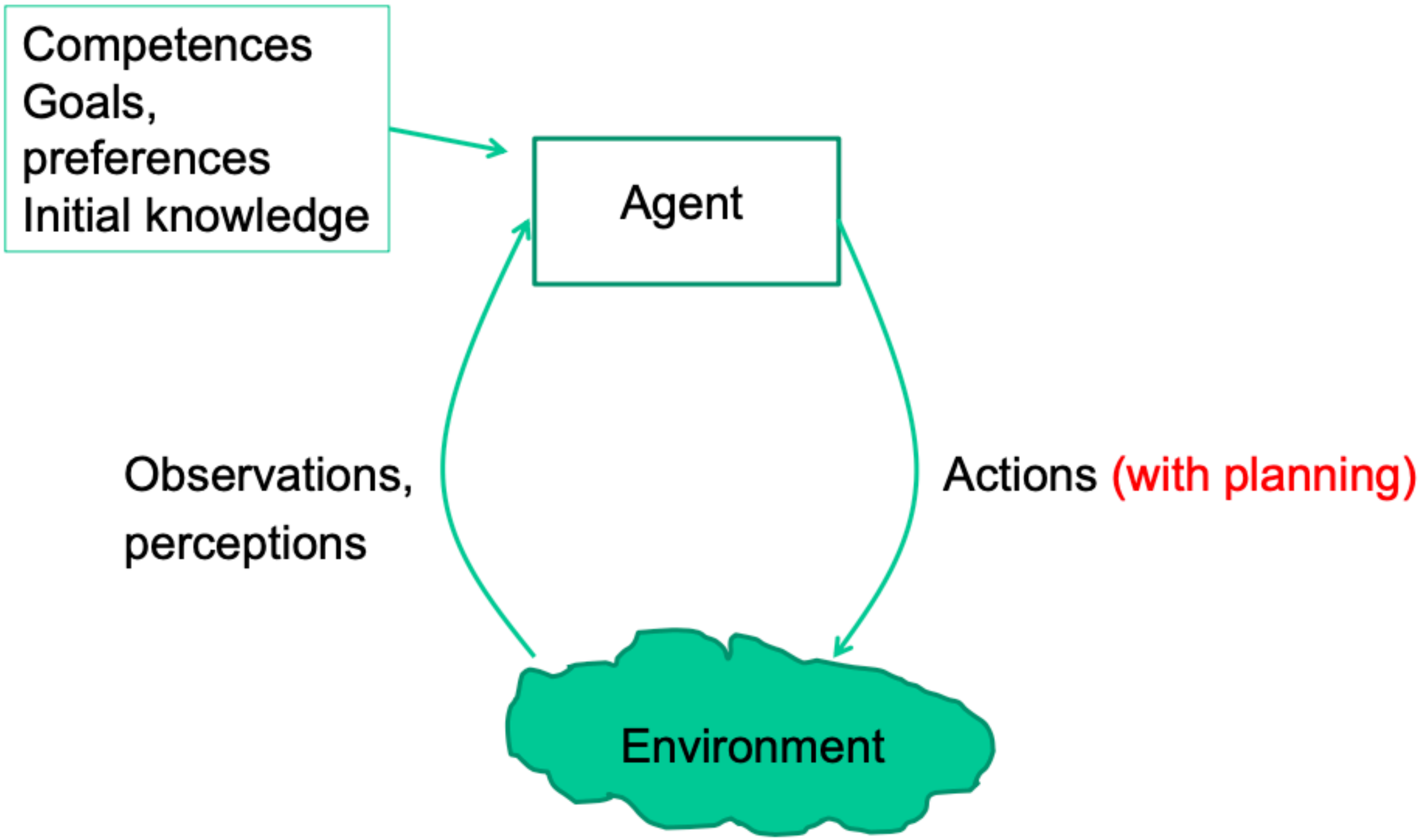
COMP3411/9814: Artificial Intelligence

Planning

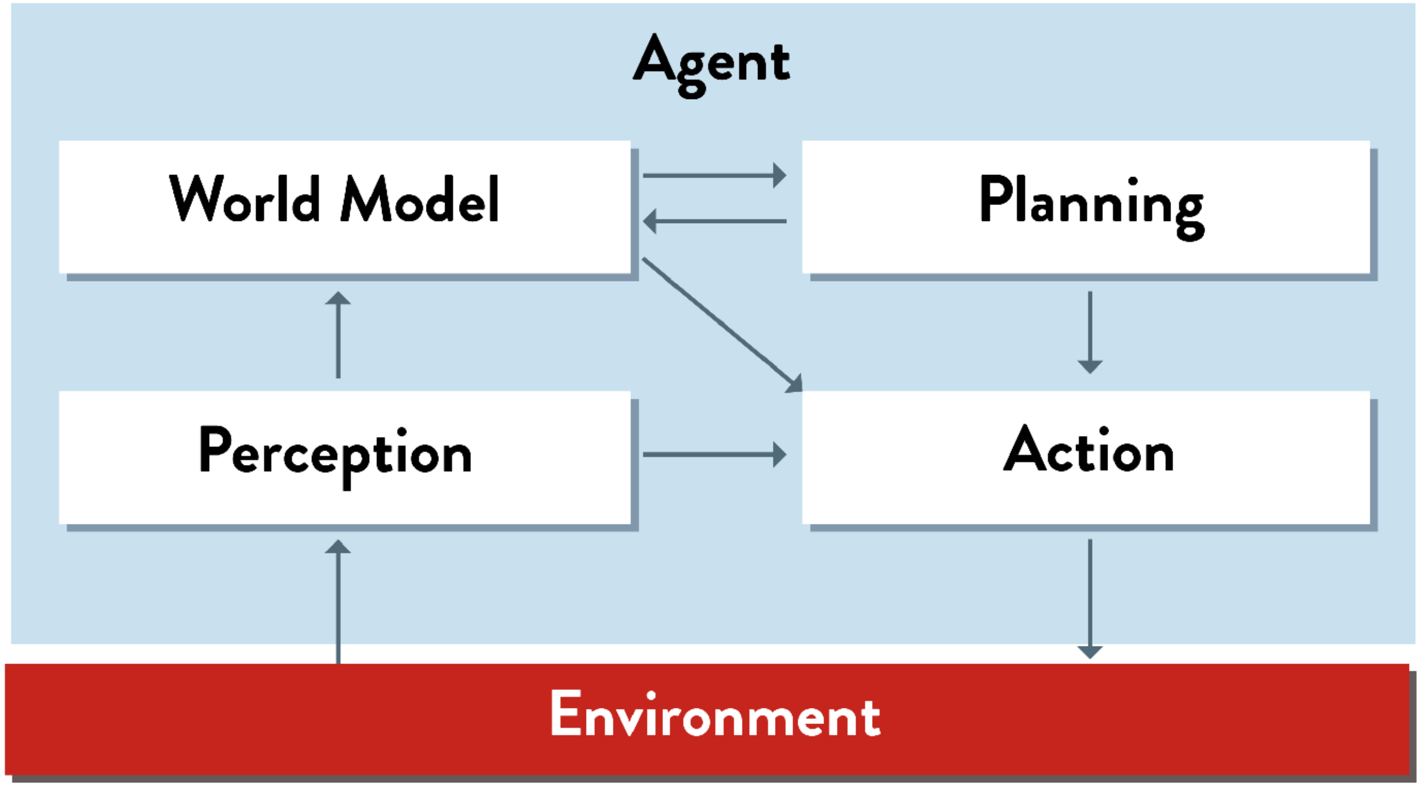
Lecture Overview

- Reasoning About Action
- STRIPS Planner
- Forward planning
- Regression Planning
- GraphPlan
- Planning as Constraint Satisfaction

Agent acting in its environment



Planning Agent

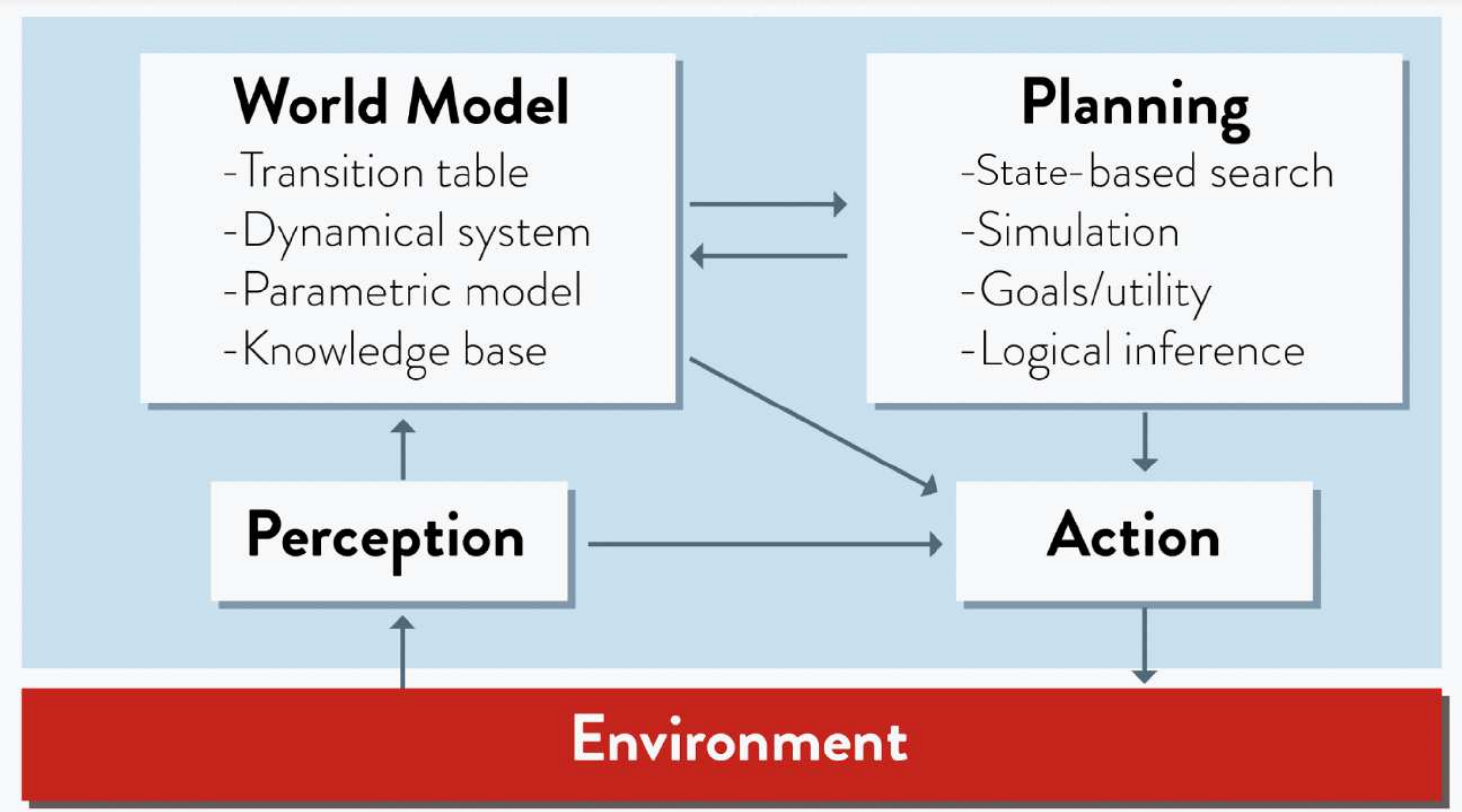


Goal-Based Agent

Planning Agent

- Decision making of this kind is fundamentally different from the condition– action rules
- It involves consideration of the future
  - “What will happen if I do such-and-such?” and
  - “Will that make me happy?”
    - In the reflex agent designs, this information is not explicitly represented

Models and Planning



Agent Plans Actions To Achieve Desired Goals

- “PLANNING” in general sense includes problem solving in state space

- “PLANNING” in narrow sense is: “means ends planning”

Planning

- Planning is deciding what to do based on an agent's ability, its goals. and the state of the world.
- Planning is finding a sequence of actions to solve a goal. Initial assumptions:
  - The world is deterministic.
  - There are no exogenous events outside of the control of the robot that change the state of the world.
  - The agent knows what state it is in.
  - Time progresses discretely from one state to the next.
  - Goals are predicates of states that need to be achieved or maintained.

Planning Agent

- The planning agent or goal-based agent is more flexible because the knowledge that supports its decisions is represented explicitly and can be modified.

- The agent's behavior can easily be changed.

Planning Agent

- Environment changes due to the performance of actions
- Planning scenario
  - Agent can control its environment
  - Only atomic actions, not processes with duration
  - Only single agent in the environment (no interference)
  - Only changes due to agent executing actions (no evolution)

- More complex examples
  - Robocup dog
  - Delivery robot
  - Self-driving car

Representation

- How to represent a classical planning problem?

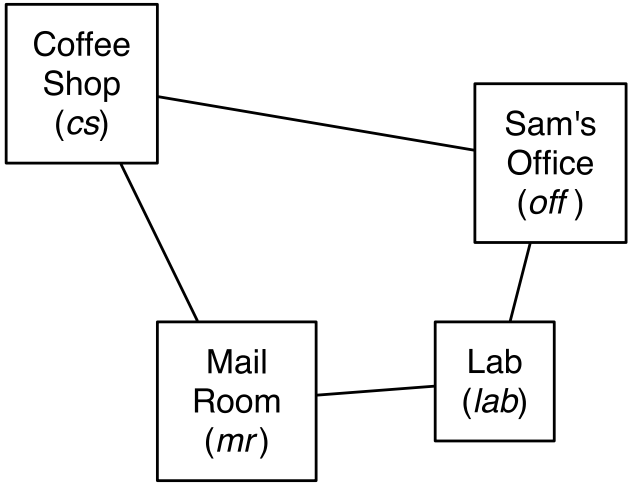
Representation

- How to represent a classical planning problem?
- Representing with States, Actions, and Goals

Actions

- A deterministic action is a partial function from states to states.
- The preconditions of an action specify when the action can be carried out.
- The effect of an action specifies the resulting state.

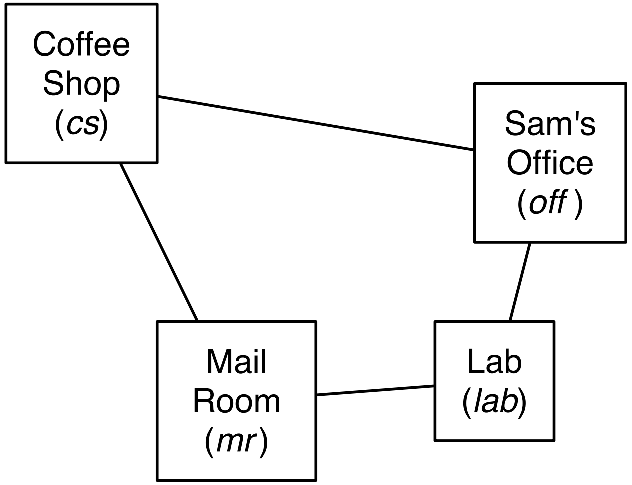
Delivery Robot Example



The delivery robot domain

The robot, called Rob, can buy coffee at the coffee shop, pick up mail in the mail room, move, and deliver coffee and/or mail.

Delivery Robot Example



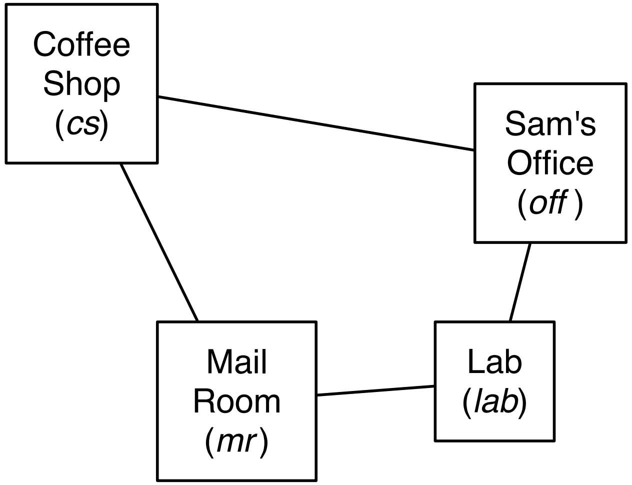
Features:

RLoc – Rob's location
RHC – Rob has coffee
SWC – Sam wants coffee
MW – Mail is waiting
RHM – Rob has mail

Actions:

mc – move clockwise
mcc – move counterclockwise
puc – pickup coffee
dc – deliver coffee
pum – pickup mail
dm – deliver mail

Delivery Robot Example



Features:
RLoc – Rob's location
RHC – Rob has coffee
SWC – Sam wants coffee
MW – Mail is waiting
RHM – Rob has mail

Features to describe states

Actions:
mc – move clockwise
mcc – move counterclockwise
puc – pickup coffee
dc – deliver coffee
pum – pickup mail
dm – deliver mail

Robot actions

State description

The state is described in terms of the following features:

- RLoc - the robot's location, which is one of the coffee shop (*cs*), Sam's office (*off*), the mail room (*mr)* or in or the laboratory (*lab*)
- SWC - Sam wants coffee. The atom *swc* means Sam wants coffee and ¬ *swc* means Sam does not want coffee.

Robot Actions

- Rob has six actions
  - Rob can move clockwise (*mc*)
  - Rob can move counterclockwise (*mcc*) or (*mac*), for now we use (*mcc*).
  - Rob can pick up coffee if Rob is at the coffee shop.
    - *puc* mean that Rob picks up coffee.
  - …..
- Assume that it is only possible for Rob to do one action at a time.

Explicit State-space Representation

Explicit State-space Representation

- The states are specifying the following:
  - the robot's location,
  - whether the robot has coffee,
  - whether Sam wants coffee,
  - whether mail is waiting,
  - whether the robot is carrying the mail.

$$\langle lab, \neg rhc, swc, \neg mw, rhm \rangle$$

Explicit State-space Representation

State	*Action*	*Resulting State*
$\langle lab, \neg rhc, swc, \neg mw, rhm \rangle$	mc	$\langle mr, \neg rhc, swc, \neg mw, rhm \rangle$
$\langle lab, \neg rhc, swc, \neg mw, rhm \rangle$	mcc	$\langle off, \neg rhc, swc, \neg mw, rhm \rangle$
$\langle off, \neg rhc, swc, \neg mw, rhm \rangle$	dm	$\langle off, \neg rhc, swc, \neg mw, \neg rhm \rangle$
$\langle off, \neg rhc, swc, \neg mw, rhm \rangle$	mcc	$\langle cs, \neg rhc, swc, \neg mw, rhm \rangle$
$\langle off, \neg rhc, swc, \neg mw, rhm \rangle$	mc	$\langle lab, \neg rhc, swc, \neg mw, rhm \rangle$
...	...	...

The complete representation includes the transitions for the other 62 states.

Explicit State-space Representation

This is not a good representation:

- There are usually too many states to represent, to acquire, and to reason with.
- Small changes to the model mean a large change to the representation.
  - Adding another feature means changing the whole representation.
- It does not represent the structure of states;
  - there is much structure and regularity in the effects of actions that is not reflected in the state transitions.

STRIPS language for problem definition

- STRIPS=Stanford Research Institute Problem Solver

- STRIPS–traditional representation
  “STRIPS-like representation”

- STRIPS makes some simplifications:
  - no variables in goals
  - positive relations given only
  - unmentioned relations are assumed false (c.w.a. – closed world assumption)
  - effects are conjunctions of relations

STRIPS Representation

- Divide the features into:
  - primitive features
  - derived features. There are rules specifying how derived can be derived from primitive features.
- For each action:
  - precondition that specifies when the action can be carried out.
  - effect a set of assignments of values to primitive features that are made true by this action.

STRIPS assumption: every primitive feature not mentioned in the effects is unaffected by the action.

Example STRIPS representation

Pick-up coffee (puc):

- precondition: [cs,¬rhc]
- effect: [rhc]

Deliver coffee (dc):

- precondition: [off,rhc]
- effect: [¬rhc,¬swc]

Feature-based representation of actions

- STRIPS is an action-centric representation

- A feature-centric representation is more flexible, as it allows for conditional effects, and non-local effects.

Feature-based representation of actions

- For each action:
  - precondition is a proposition that specifies when the action can be carried out.
- For each feature:
  - causal rules that specify when the feature gets a new value and
  - frame rules that specify when the feature keeps its value.

Example feature-based representation

- Precondition of pick-up coffee (puc):

  $RLoc=cs \land \neg rhc$

- Rules for location is cs:

  $RLoc'=cs \leftarrow Rloc = off \land Act=mcc$

  $RLoc'=cs \leftarrow Rloc = mr \land Act=mc$

  $RLoc'=cs \leftarrow Rloc = cs \land Act \neq mcc \land Act \neq mc$

- Rules for "robot has coffee"

  $rhc' \leftarrow rhc \land Act \neq dc$

  $rhc' \leftarrow Act=puc$

Example feature-based representation

- Precondition of pick-up coffee (puc):

  $RLoc=cs \land \neg rhc$

- Rules for location is cs:

  causal rules

  $RLoc'=cs \leftarrow Rloc = off \land Act=mcc$

  $RLoc'=cs \leftarrow Rloc = mr \land Act=mc$

  frame rule

  $RLoc'=cs \leftarrow Rloc = cs \land Act \neq mcc \land Act \neq mc$

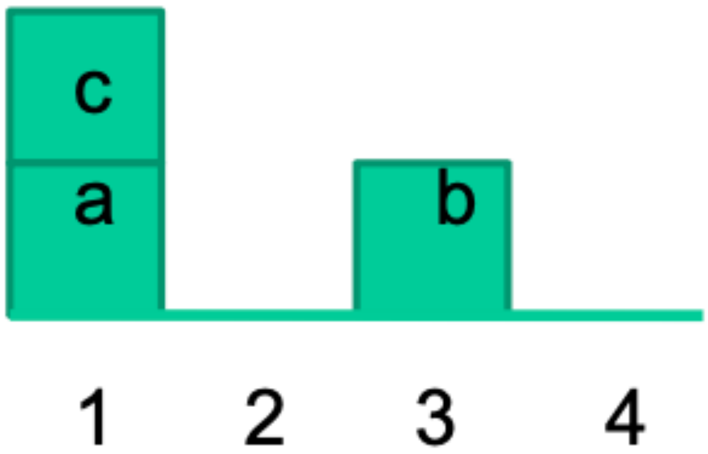
- Rules for "robot has coffee"

  $rhc' \leftarrow rhc \land Act \neq dc$

  $rhc' \leftarrow Act=puc$

States are Represented with Relations

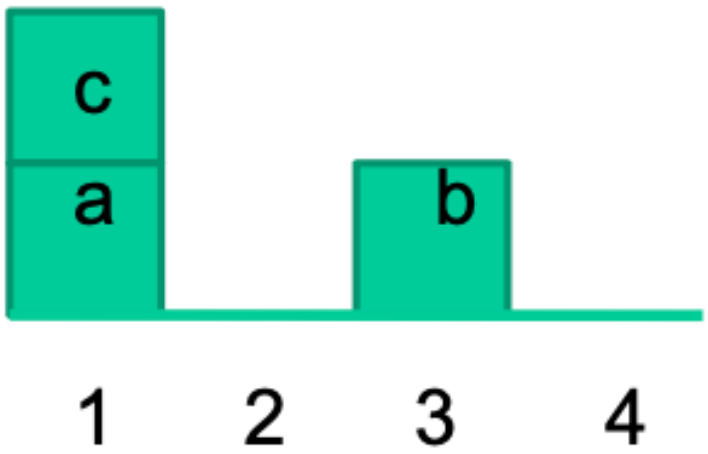
Example state from blocks world



This state can be represented by the following set of relations:

States are Represented with Relations

Example state from blocks world



This state can be represented by the following set of relations:

on(c,a), on(a,1), on(b,3), clear(2), clear(4), clear(b), clear(c)

Defining Goals and possible Actions

- Example of goals:

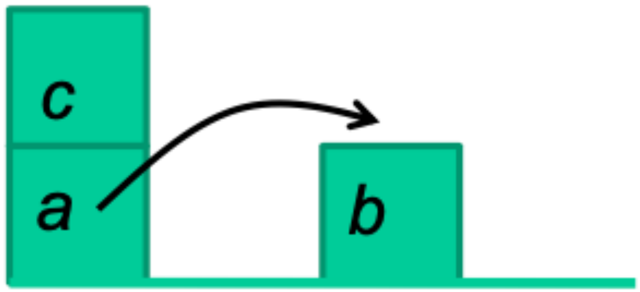
  on(a,b), on(b,c)

- Example of action:

  move(a, 1, b)

  (Move block a from 1 to b)

- Action preconditions:



Defining Goals and possible Actions

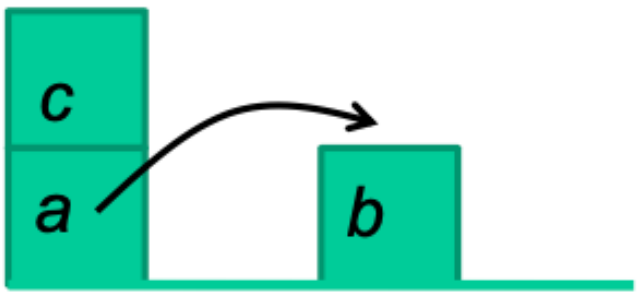
- Example of goals:

  on(a,b), on(b,c)

- Example of action:

  move(a, 1, b)

  (Move block a from 1 to b)



- Action preconditions:

  clear(a), on(a,1), clear(b)

"add" (true after action)

- Action effects:

  on(a,b), clear(1), ~on(a,1), ~clear(b)

"delete" (no longer true after action)

(~ = ¬)

Action Schema

- Action schema represents a set of actions using variables (variable names here written with capital initials)

move(X, Y, Z)

X is any block

Y and Z are any block or location

- Precondition: **on(X,Y),clear(X),clear(Z)**
- Adds: **on(X,Z),clear(Y)**
- Deletes: **on(X,Y),clear(Z)**

STRIPS language for problem definition

- STRIPS makes some simplifications:
  - no variables in goals
  - positive relations given only
  - unmentioned relations are assumed false (c.w.a. – closed world assumption)
  - effects are conjunctions of relations

ADL - Action Description Language

- ADL removes some of the STRIPS assumptions, for example:

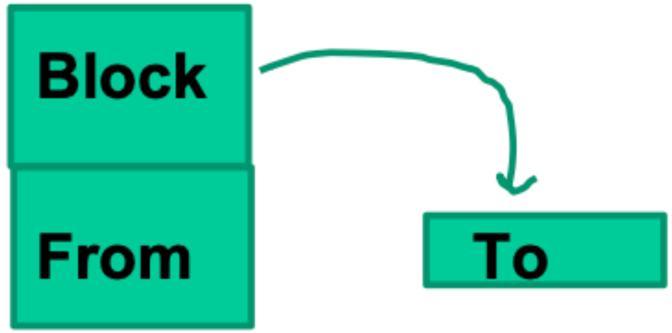
STRIPS	ADL
States: + literals only **on(a,b), clear(a)**	**on(a,b), clear(a), ~clear(b)**
Effects: + literals only Add **clear(b)**, Delete **clear(c)**	Add **clear(b)** and **~clear(c)** Delete **~clear(b)** and **clear(c)**
Goals: no variables **on(a,c), clear(a)**	**Exists X: on(X,c), clear(X)**

STRIPS Representation

- Divide the features into:
  - primitive features
  - derived features. There are rules specifying how derived can be derived from primitive features.
- For each action:
  - precondition that specifies when the action can be carried out.
  - effect a set of assignments of values to primitive features that are made true by this action.

STRIPS assumption: every primitive feature not mentioned in the effects is unaffected by the action.

Domain Specification For Blocks World



Action:

move(Block, From, To)

Action precondition:

clear(Block), clear(To), on(Block, From)

Positive effects (“add“):

on(Block,To), clear(From)

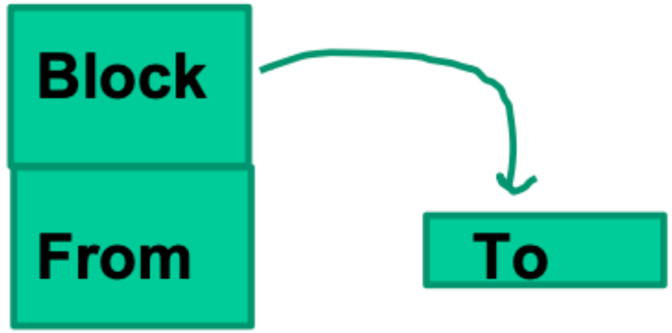
Negative effects (“del”)

on(Block,From), clear(To)

Better With Additional Constraints

Action:

move(Block, From, To)



Precondition for Action:

clear(Block), clear(To), on(Block, From)

Additional constraints:

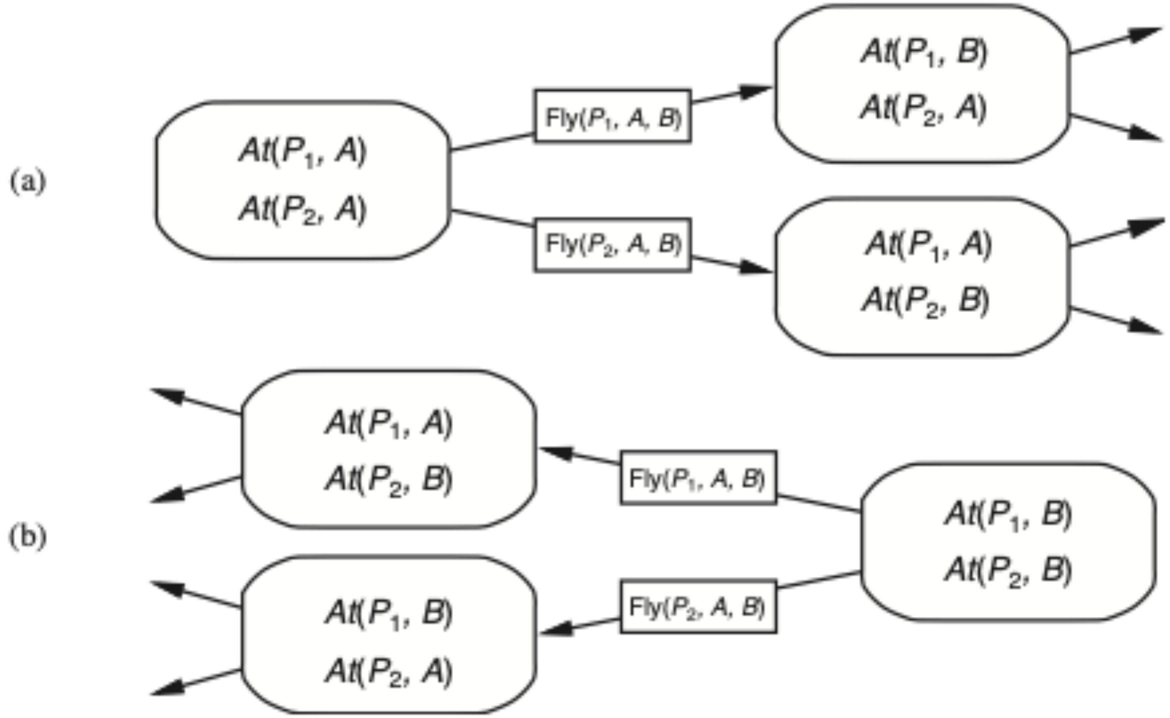
block(Block),	% Object Block to be moved must be a block
object(To),	% "To" is an object, i.e. a block or a place
To $\neq$ Block,	% Block cannot be moved to itself
object(From),	% "From" is a block or a place
From $\neq$ To,	% Move to new position
Block $\neq$ From	

Planning

- Plan — sequence (or ordered set) of actions to achieve some goal
- Planner — problem solver that produces plans
- Goal – typically a conjunction of literals
- Initial State – typically a conjunction of literals
- Blocks World Example for goal $on(B, C) \wedge on(C, Table)$
  - $move(C, A, Table), move(B, Table, C)$

Simple Planning Algorithms

- Forward search and goal regression



Problem with forward search is state space can be very large
Problem with regression is that it is hard and doesn't always work

Forward Search with Plan Graphs

- Only consider “propositional” plans
  - *Si* contains all literals that *could* hold at time *I*
  - *Ai* contains all actions that *could* have preconditions satisfied at time *i*
  - Actions linked to preconditions
  - Literals that persist from time *i* to time *i* + 1 linked via actions
  - Mutual exclusion (mutex) links between actions/literals at same time

Mutual Exclusion

- Actions
  - Inconsistent effects: One action negates an effect of the other
  - Competing needs: Precondition of one action is mutually exclusive with a precondition of the other
- Literals
  - One literal is the negation of the other
  - Inconsistent support: Each possible pair of actions that could achieve the two literals is mutually exclusive

Specification Of Blocks And Locations

% Our blocks world: three blocks a, b and c, and 4 locations

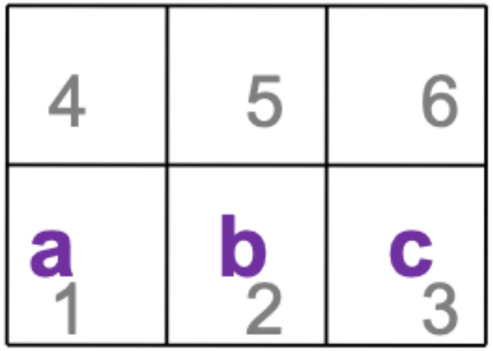
block(a). block(b). block(c).

place(1). place(2). place(3). place(4).

% X is an object if X is a block or a place

object(X) ← (block(X) V place(X))

Robots On Grid In Strips



Robots: a, b, c, cells 1, ..., 6

Goal: at(a,3)

Plan: m(b,2,5) ⟶ m(a,1,2) ⟶ m(c,3,6) ⟶ m(a,2,3)

GraphPlan Algorithm

Graph = Initial plan graph with initial state S_0

nogoods = empty set

For $t = 0, \cdots$

- If all goals are non-mutex in S_t
  - Extract solution from graph
    - Graph as CSP with variables T/F for when action in the plan
    - Or heuristically guided regression from S_t to S_0
  - If valid solution, return solution
- If graph and nogoods didn't change then return failure
- Expand graph to next level

GraphPlan Expansion Step

Add actions to A_i whose preconditions are at S_i

Add "persistence actions" to A_i for literals from S_i

Add mutex links to A_i for actions that cannot occur together

Add effects of all actions in A_i to S_{i+1}

Add literals to S_{i+1} for persistence actions from A_i

Add mutex links to S_{i+1} for literals that cannot occur together

Forward Search with Plan Graphs

- A forward planner searches the state-space graph from the initial state looking for a state that satisfies a goal description.

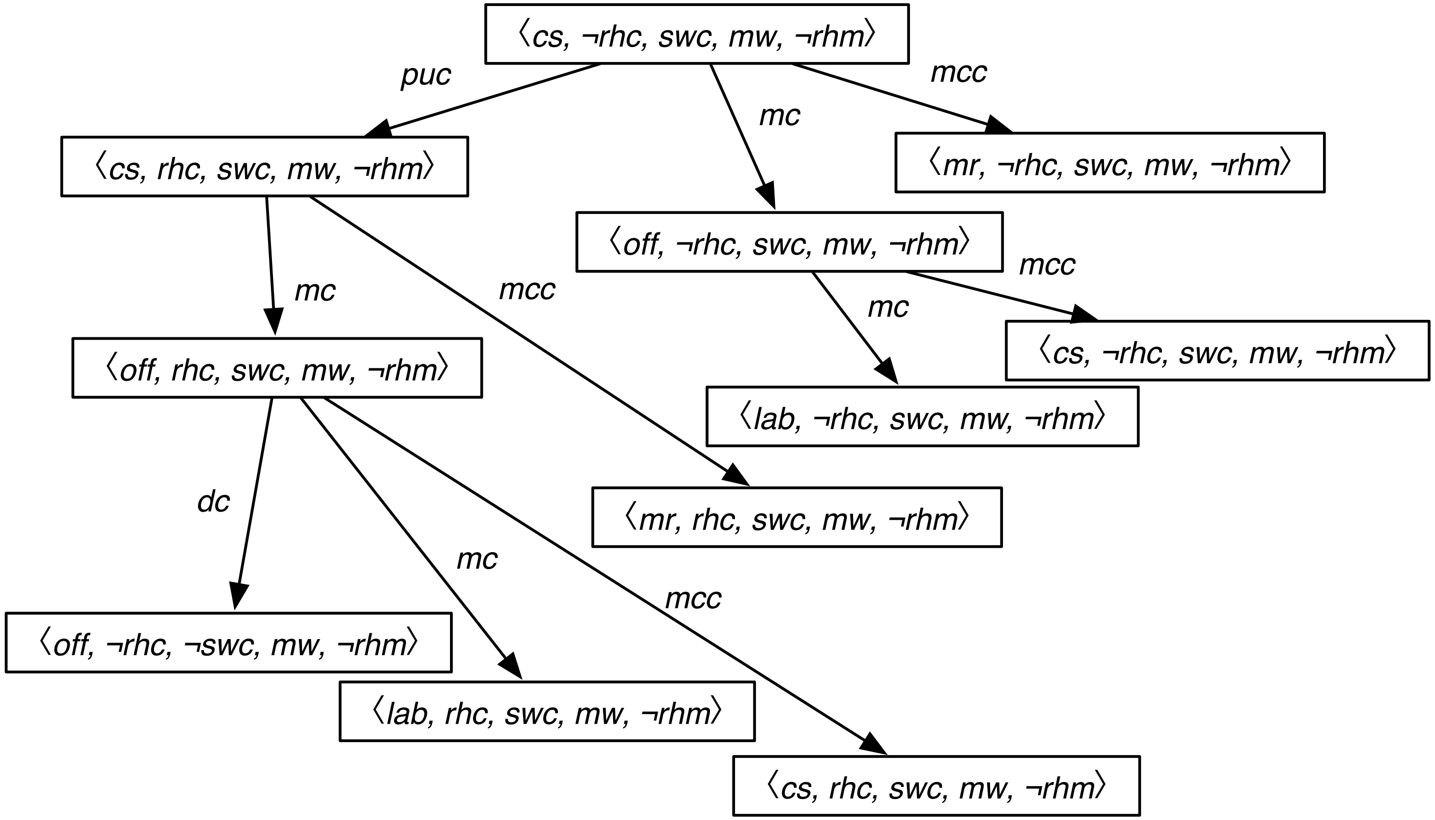
  - It can use any of the search strategies

Forward Search with Plan Graphs

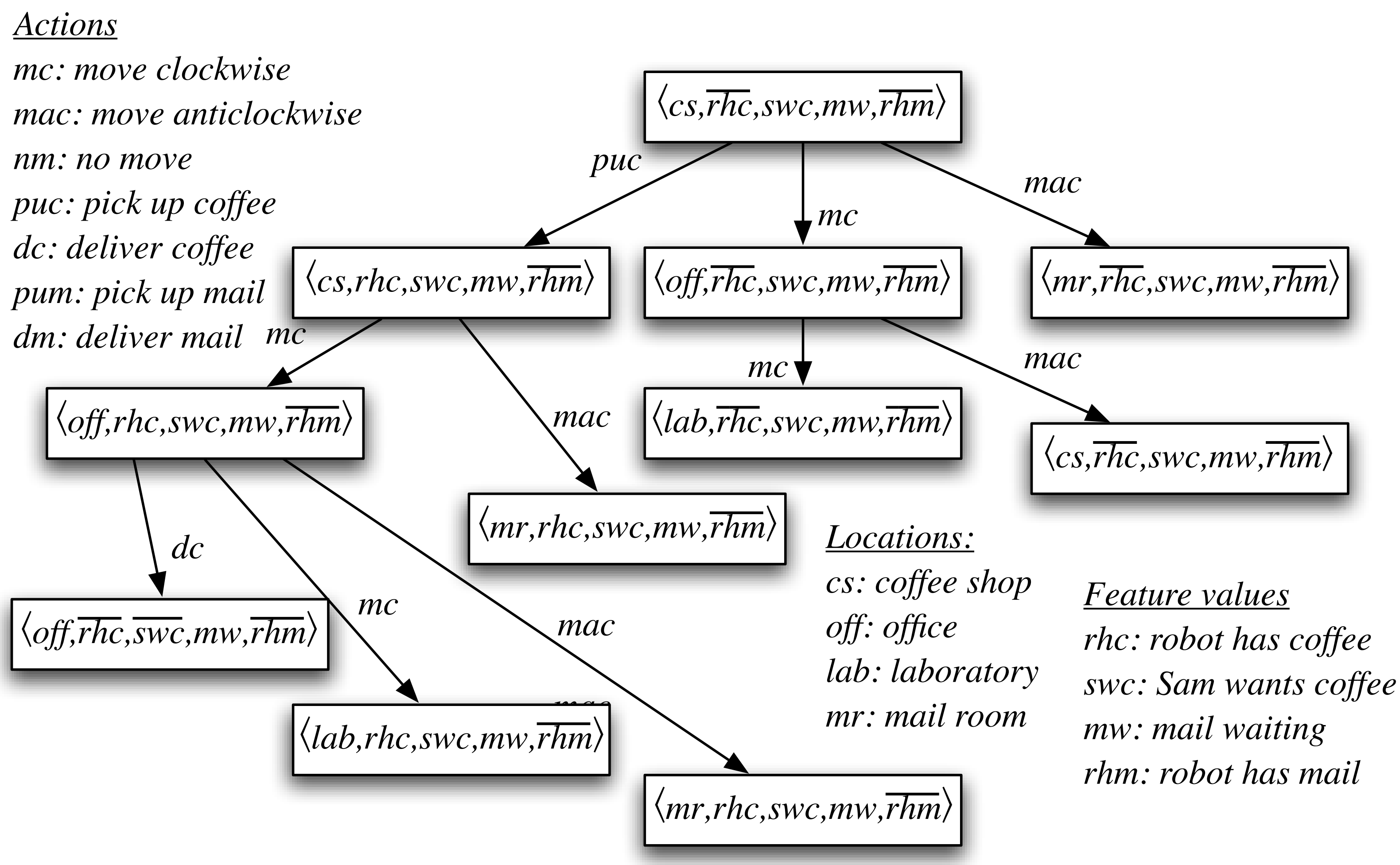
The search graph is defined as follows:

- The nodes are states of the world, where a state is a total assignment of a value to each feature.
- The arcs correspond to actions.
- The start node is the initial state.
- The goal condition for the search, *goal(s)* is true if state ss satisfies the achievement goal.
- A path corresponds to a plan that achieves the goal.

Forward Search with Plan Graphs



Forward Search with Plan Graphs

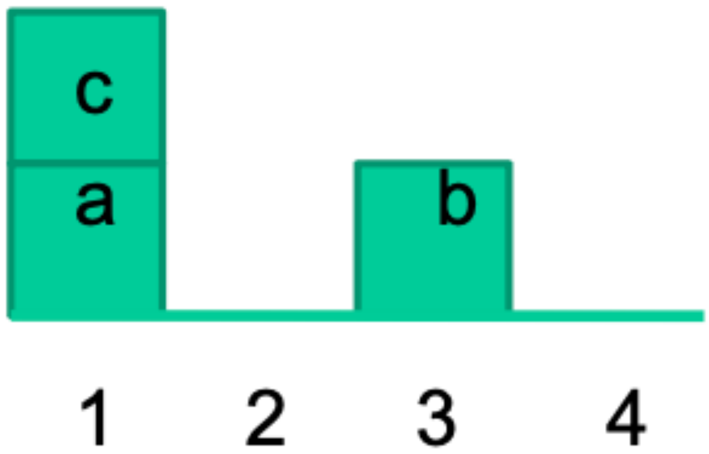


Forward Search with Plan Graphs

- Using a forward planner is not the same as making an explicit state-based representation of the actions

- The relevant part of the graph is created dynamically from the representations of the actions.

Principle of means-ends analysis (Optional)



In this state, the following relations hold:

on(c,a), on(a,1), on(b,3), clear(2), clear(4), clear(b), clear(c)

Let goal of plan be **on(a,b)**; find plan:

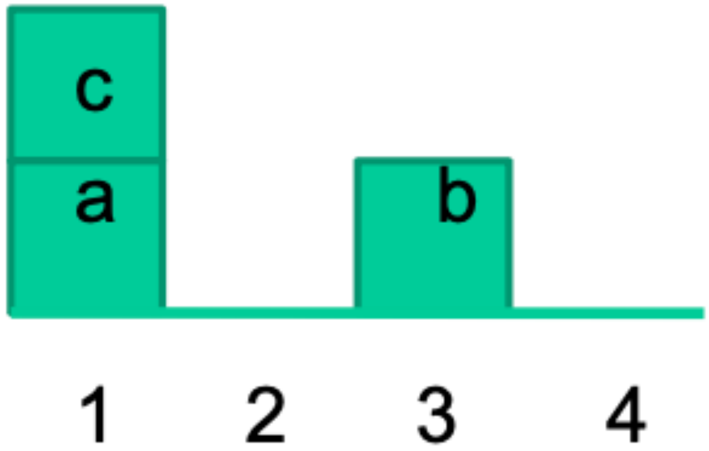
What action establishes **on(a,b)**? Such action is: move(a,X,b)

What is precondition COND for this action?

COND: **on(a,X), clear(a), clear(b)**

Set intermediate goal COND, find plan for COND Etc.

Principle of means-ends analysis (Optional)



In this state, the following relations hold:

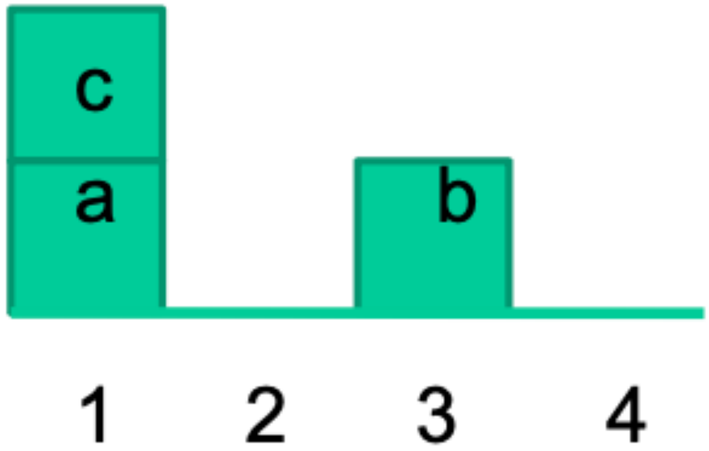
on(c,a), on(a,1), on(b,3), clear(2), clear(4), clear(b), clear(c)

Let goal of plan be **on(a,b)**; find plan:

What action establishes **on(a,b)**? Such action is: move(a,X,b)

What is precondition COND for this action?

Principle of means-ends analysis (Optional)



In this state, the following relations hold:

on(c,a), on(a,1), on(b,3), clear(2), clear(4), clear(b), clear(c)

Let goal of plan be **on(a,b)**; find plan:

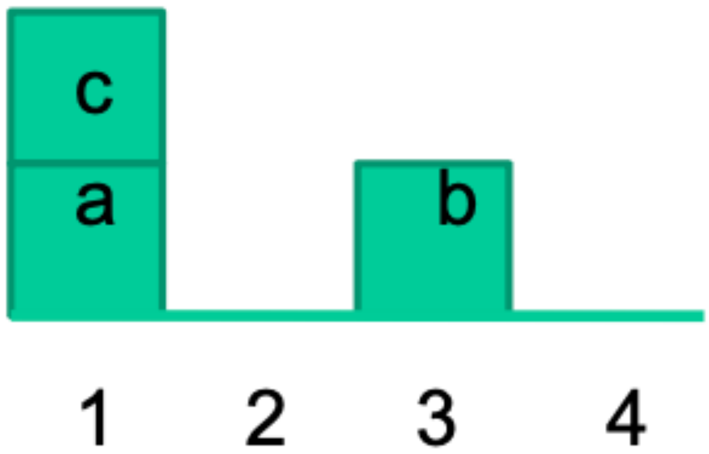
What action establishes **on(a,b)**? Such action is: move(a,X,b)

What is precondition COND for this action?

COND: **on(a,X), clear(a), clear(b)**

Set intermediate goal COND, find plan for COND Etc.

Principle of means-ends analysis (Optional)



In this state, the following relations hold:

on(c,a), on(a,1), on(b,3), clear(2), clear(4), clear(b), clear(c)

Let goal of plan be **on(a,b)**; find plan:

What action establishes **on(a,b)**? Such action is: move(a,X,b)

What is precondition COND for this action?

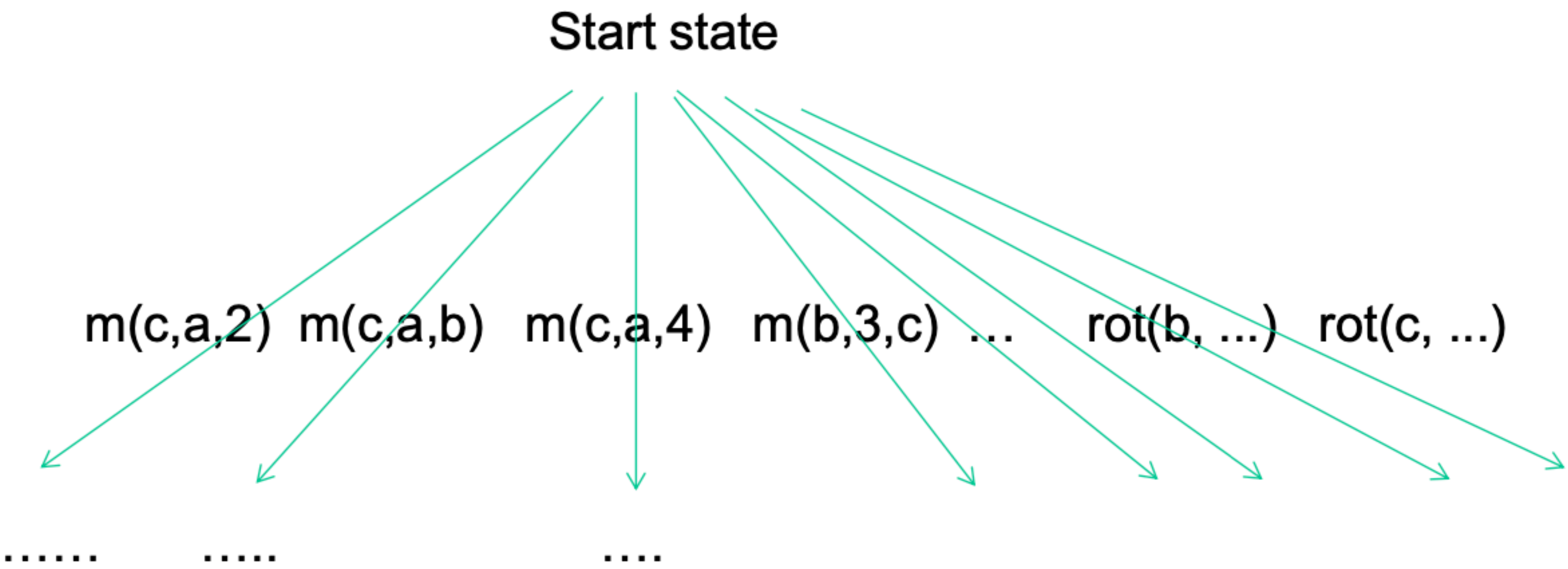
COND: **on(a,X), clear(a), clear(b)**

Set intermediate goal COND, find plan for COND Etc.

Comparison with state space (Optional)

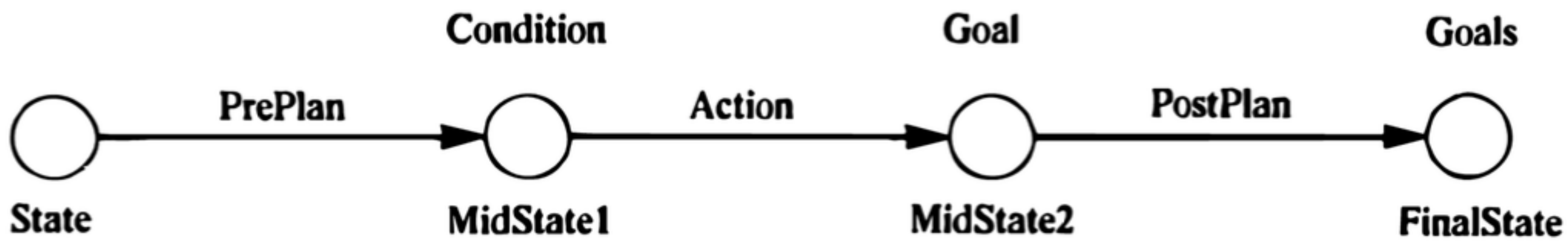
- Instate-space: search state space
- In means-ends planning: search space of sets of goals
- Space of sets of goals = abstraction of state space
- What is better? Means-ends planning may be able to avoid searching useless actions, see example on next slide

Actions in state space (Optional)



Means-ends planning in Strips (Optional)

One possible realisation of means-ends planning



Nondeterministic strips algorithm

```
procedure plan( InitialState, Goals, Plan, FinalState)
  if Goals ⊆ InitialState then Plan = [ ] else          % All goals achieved
  begin
     Select a goal G from Goals;
     Select an action A that achieves G;                 % adds( A, G)
     PreCond = preconditions of A;
     plan( InitialState, PreCond, PrePlan, MidState1) ;  % Enable A
     Apply A to MidState1 giving MidState2;
     plan( MidState2, Goals, PostPlan, FinalState);      % Achieve remaining goals
     Plan = concatenate( PrePlan, [ Action ], PostPlan)
  end
```

Strips in Prolog

```
% plan( State, Goals, Plan, FinalState)
plan( State, Goals, [ ], State) :-
    satisfied( State, Goals).

plan( State, Goals, Plan, FinalState) :-
    conc( PrePlan, [Action | PostPlan], Plan),      % Divide plan
    select( State, Goals, Goal),                    % Select a goal
    achieves( Action, Goal)                         % Relevant action
    can( Action, Condition),
    plan( State, Condition, PrePlan, MidState1),    % Enable Action
    apply( MidState1, Action, MidState2),           % Apply Action
    plan( MidState2, Goals, PostPlan, FinalState).  % Remaining goals
```

Additional details

- Search strategy (depth-first, breadth-first,...)
- Goal protection: do not destroy what you already achieved!
- But: goal protection is not always possible!

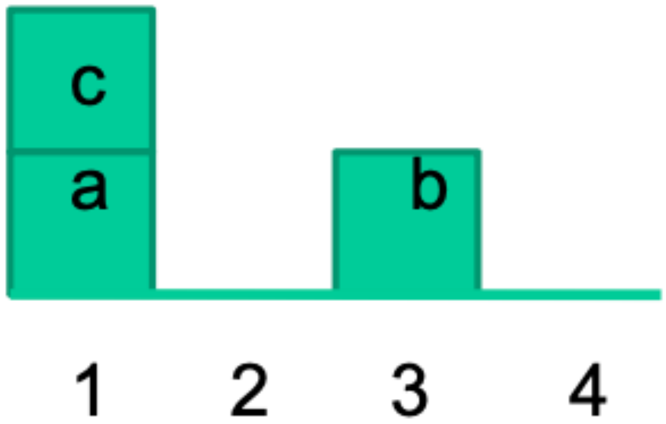
Realisation with iterative deepening

- Start with plan of length 0 and keep increasing maximal allowed length of plan, until plan is found
- On each iteration (for each maximal plan length) search all possible plans with depth-first search

Realisation with iterative deepening

- Start with plan of length 0 and keep increasing maximal allowed length of plan, until plan is found
- On each iteration (for each maximal plan length) search all possible plans with depth-first search
- Surprise is possible, for example in the case of Sussman's anomaly (on next slide)

Sussman’s anomaly



Goals: on(a,b), on(b,c)

Basic STRIPS planner with breadth-first search produces:

move(c, a, 2)

move(b, 3, a) ??? What is the point of this ???

move(b, a, c)

move(a, 1, b)

Problem is: STRIPS concentrates on solving a single goal at a time

Completeness

- Even with global iterative deepening, our planner still has problems.
- E.g.it finds a four step plan above for our example block stack
- Why STRIPS cannot find the optimal, 3-step plan? Basic STRIPS is **incomplete**! It dopes not consider all possible plans.
- Problem: locality (only work towards achieving one goal G at a time, temporarily ignoring other goals until G is achieved)
- Sometimes referred to as “linearity” (goals are achieved in “linear order”)

Goal regression

- STRIPS solves goals one after another "locally" (when solving one goal it does not consider other goals)
- Better: "global planning" (keep in mind all the goals all the time)
- One idea to achieve global planning is goal regression
- This is based on concept of "Regressing Goals through Action"

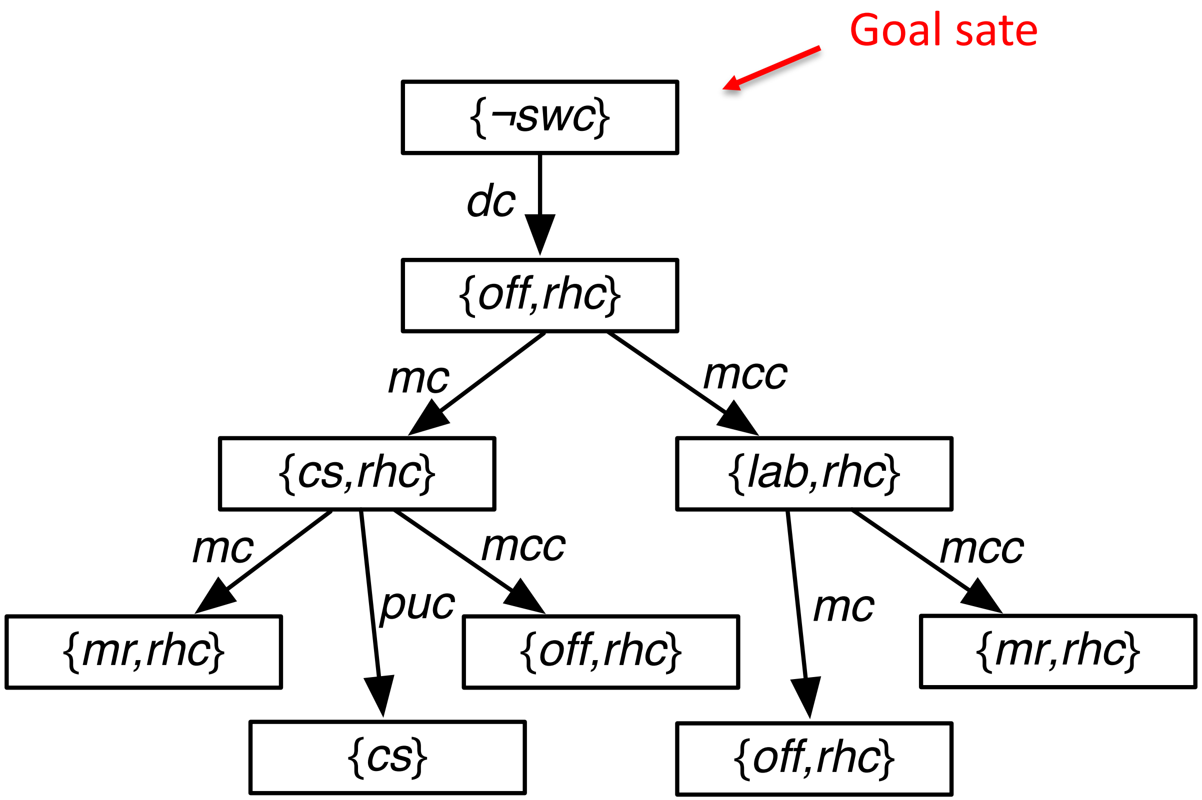
$$\text{RegressedGoals} \xrightarrow[\text{Action}]{} \text{Goals}$$

Goal regression

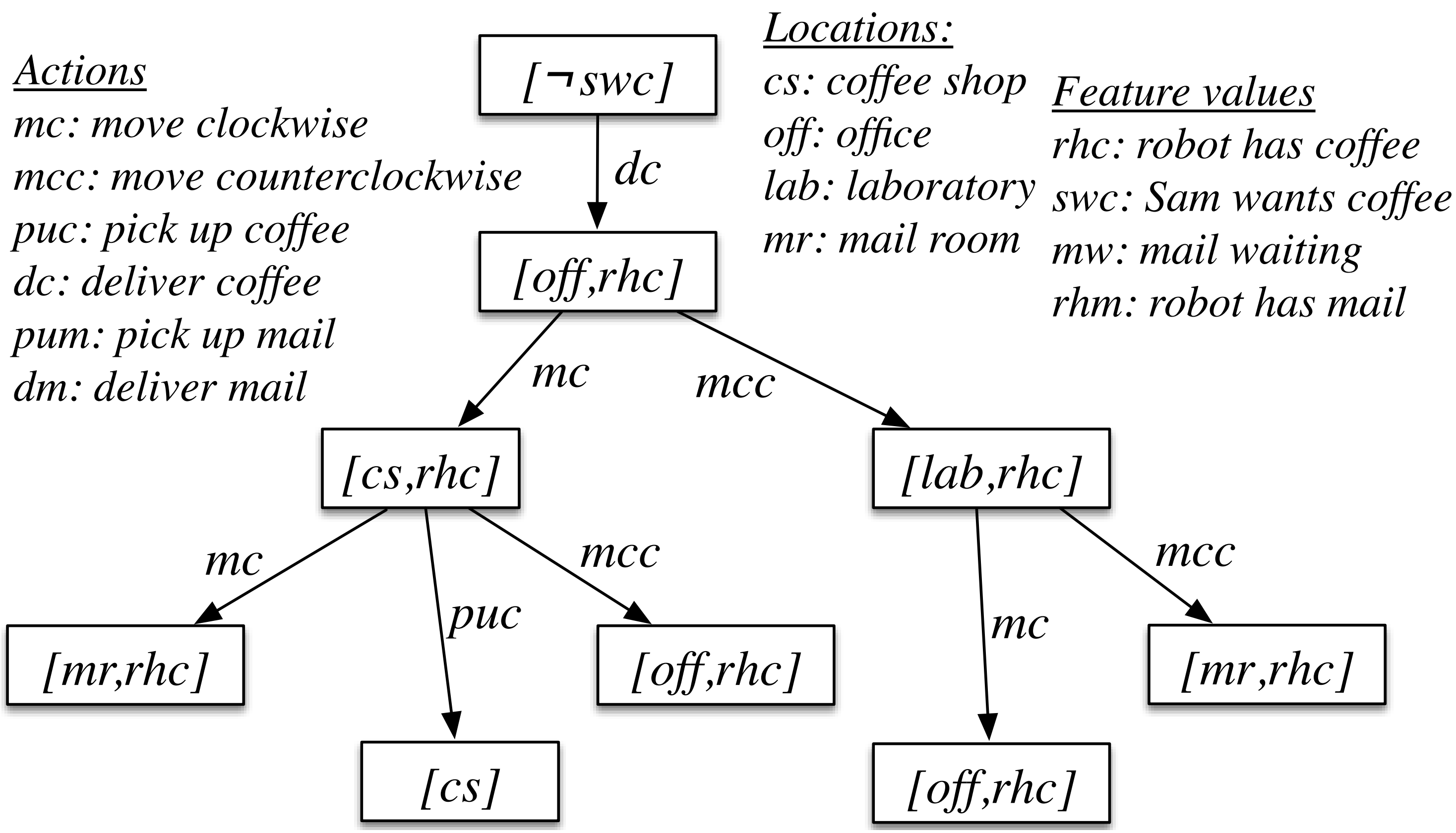
Regression planning is searching in the graph defined by the following:

- The nodes are subgoals.
- The arcs correspond to actions. An arc from node g to g', labeled with action *act*, means
  - *act* is the last action that is carried out before subgoal g is achieved, and
  - node g' is a subgoal that must be true immediately before *act* so that g is true immediately after *act*.
- The start node is the planning goal to be achieved.
- The goal condition for the search, goal(g), is true if g is true of the initial state.

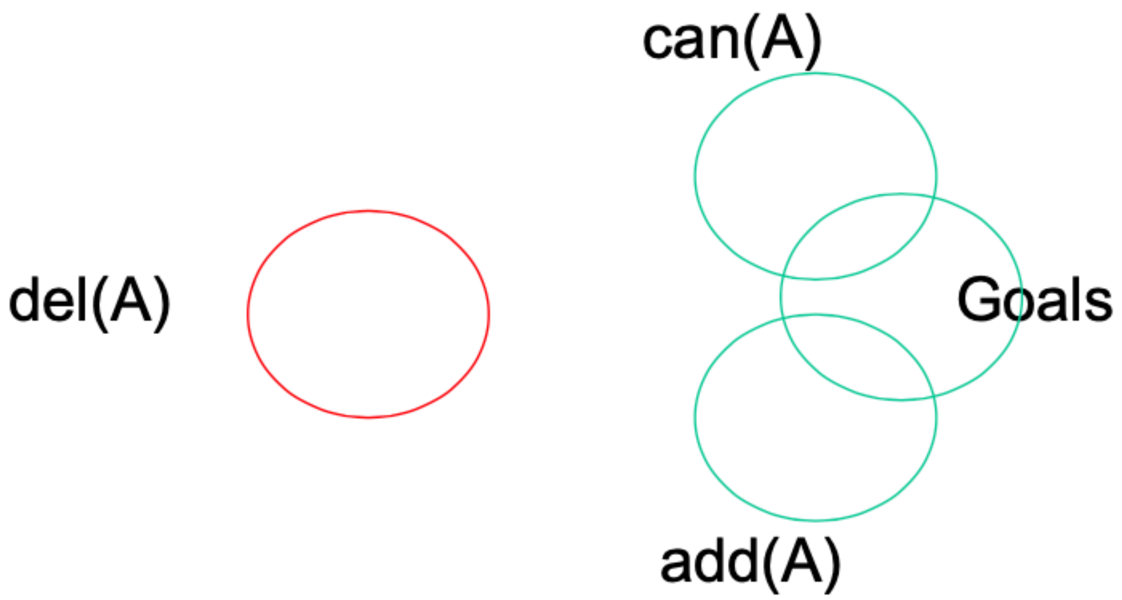
Goal regression



Goal regression



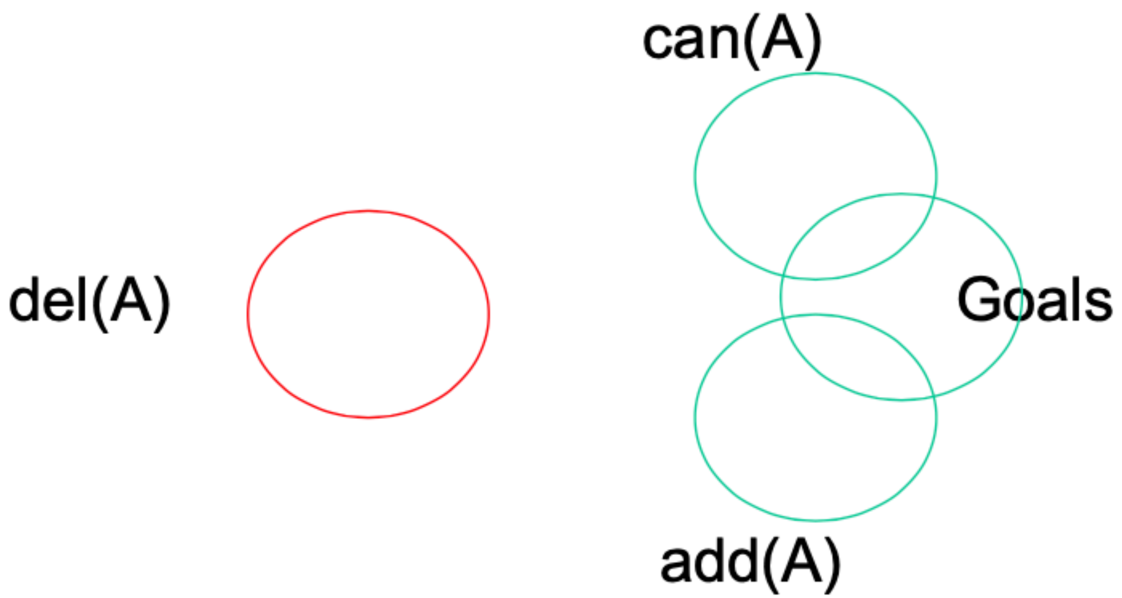
Goal regression



RegressedGoals = Goals + can(A) - add(A)

Goals and del(A) are disjoint

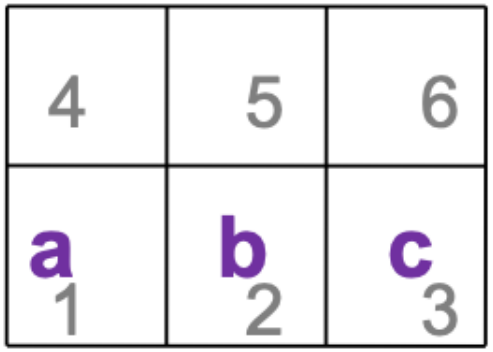
Goal regression



Goal regression enables “global” planning:

Planner can see all relevant goals at any point of planning

Robots On Grid In Strips



Robots: a, b, c,   Cells 1, ..., 6

Goal: at(a,3)

Domain definition

Action: Robot R moves from A to B

m(R,A,B)

Preconditions:

at(R,A), c(B)

Additional constraints:

robot(R), adjacent(A,B) % R is a robot, A and B are adjacent

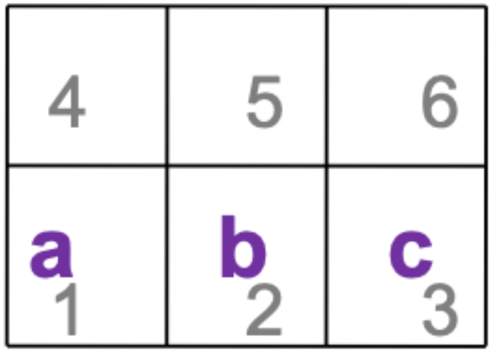
Positive effects:

at(R,B), c(A)

Negative effects:

at(R,A), c(B)

Robots On Grid In Strips



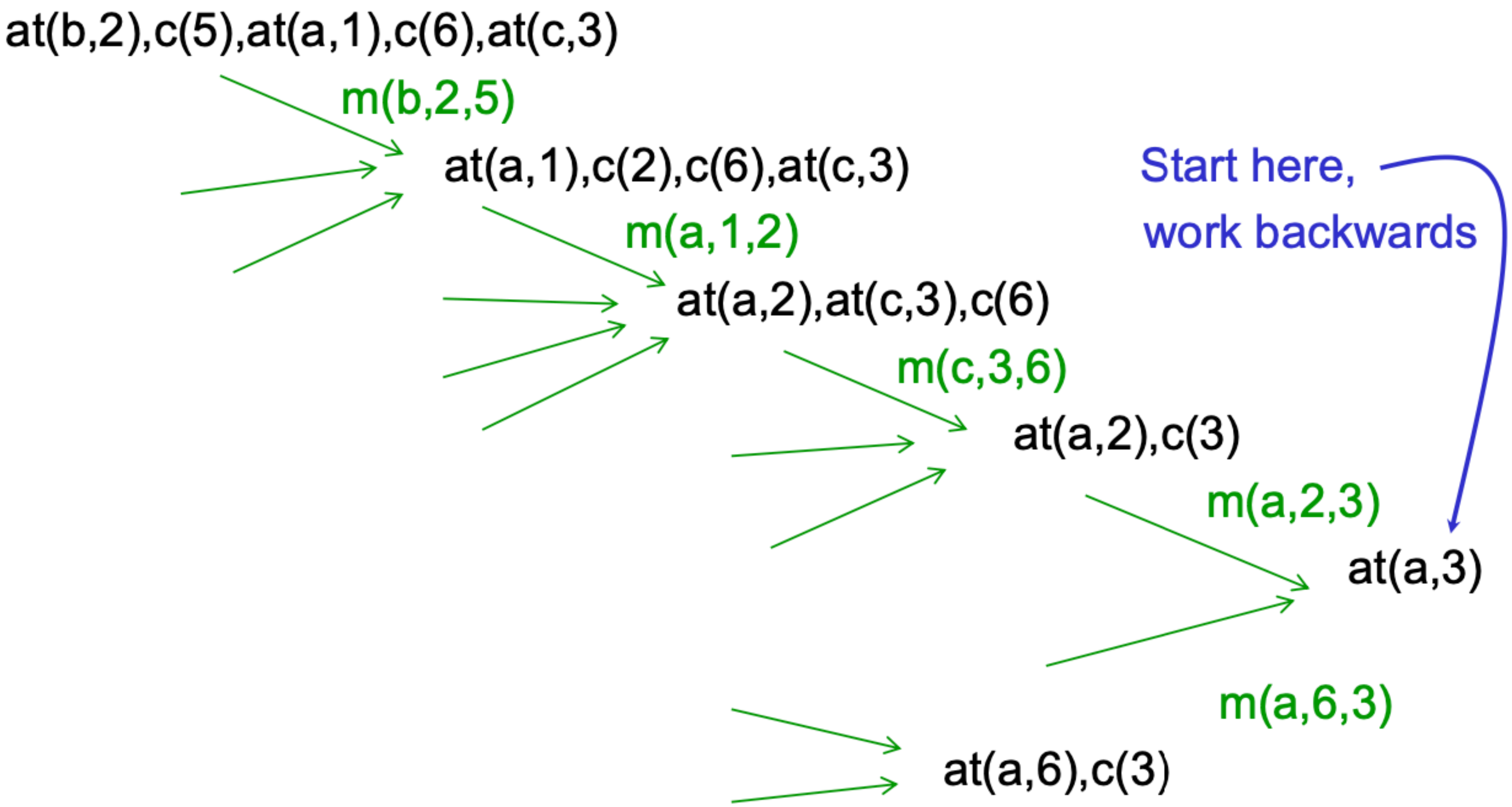
Robots: a, b, c, cells 1, ..., 6

Goal: at(a,3)

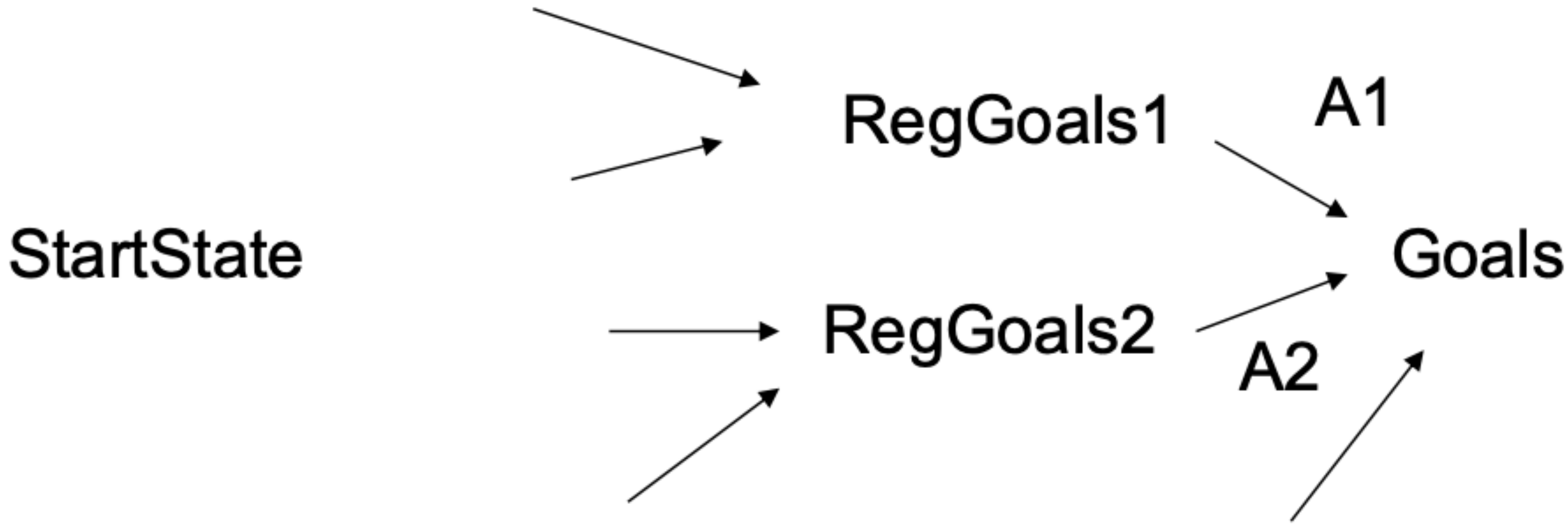
Plan: m(b,2,5) $\longrightarrow$ m(a,1,2) $\longrightarrow$ m(c,3,6) $\longrightarrow$ m(a,2,3)

Finding a plan for goal at(a,3)

Initial state: at(a,1), at(b,2), at(c,3), c(4), c(5), c(6)



State space for planning with goal regression



Goal state and heuristic

- "Goal" condition for this search space:
  RegressedGoals is a subset of StartState

- A possible heuristic function for this search space:
  #regressed goals that are not true in StartState:

$$h = | \text{RegressedGoals} - \text{StartState} |$$

Question

- Is this heuristic function for the blocks world optimistic? That is, does it satisfy the condition of admissibility theorem for best-first search?

- If not, can it be modified to become optimistic?

Planning as Constraint Satisfaction

CSP for each planning horizon k (vary k as needed)

- Variables
  - Create a variable for each literal and time $0,\cdots,k$
  - Create a variable for each action and time $0,\cdots,k-1$
- Constraints
  - State constraints: literals at time t
  - Precondition constraints: actions and states at time t
  - Effect constraints: actions at time t , literals at times t and $t + 1$
  - Action constraints: actions at time t (mutual exclusion)
  - Initial state constraints: literals at time 0
  - Goal constraints: literals at time k

The delivery robot CSP planner with factored actions

$RLoc_i$
- Rob's location

RHC_i
- Rob has coffee

SWC_i
- Sam wants coffee

MW_i
- Mail is waiting

RHM_i
- Rob has mail

$Move_i$
- Rob's move action

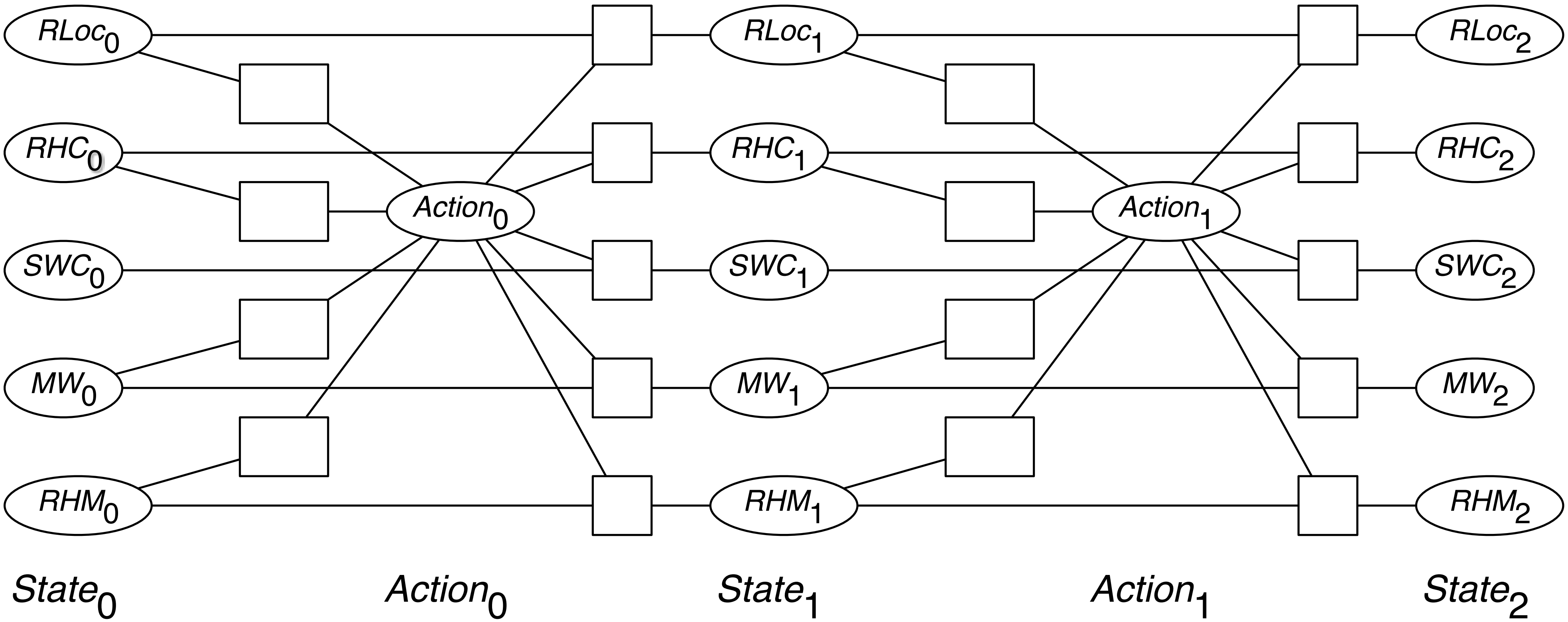
PUC_i
- Rob picks up coffee

$DelC$
- Rob delivers coffee

PUM_i
- Rob picks up mail

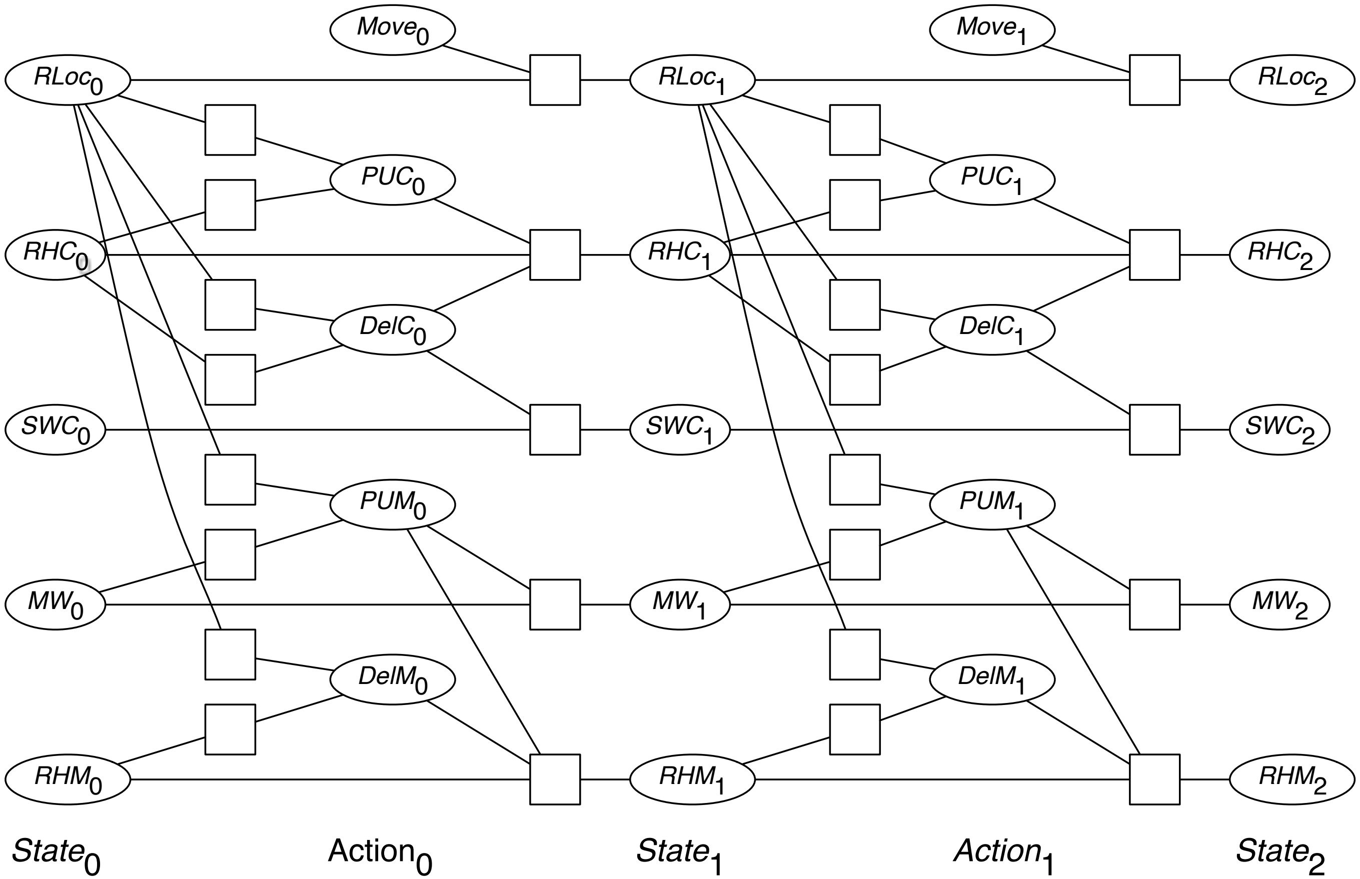
$DelM_i$
- Rob delivers mail

Planning as Constraint Satisfaction



CSP for planning horizon of 2.

Planning as Constraint Satisfaction



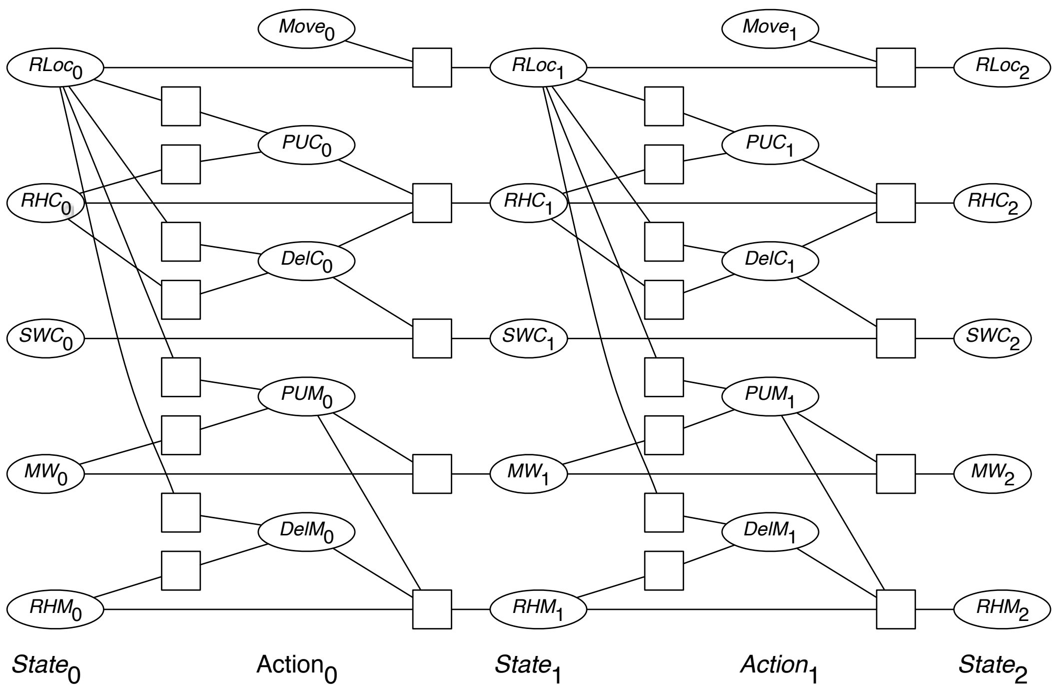
CSP for planning horizon of 2.

There can be an extra set of constraints **mutex constraints -** to specify which action features cannot co-occur.

Planning as Constraint Satisfaction

The agent can be seen as doing more than one action in a single stage.

- For some of the actions at the same stage, the robot can do them in any order, such as delivering coffee and delivering mail.
- Some of the actions at the same stage need to be carried out in a particular order, for example, the agent must move after the other actions.



Summary

- Planning is the process of choosing a sequence of actions to achieve a goal.
- An action is a partial function from a state to a state.
- Two representations for actions that exploit structure in states are
  - the STRIPS representation, which is an action-centric representation,
  - the feature-based representation of actions, which is a feature-centric representation.
  - The feature-based representation is more powerful than the STRIPS representation; it can represent anything representable in STRIPS, but can also represent conditional effects.

Summary

- A forward planner searches in the state space from the initial state to a goal state.
- A regression planner searches backwards from the goal, where each node in the search space is a subgoal to be achieved.

Summary

- Reasoning about action interesting from philosophical point of view
- Recent advances in planning give great improvements in efficiency
- Planning makes use of CSP framework with heuristics
- Multi-agent systems, dynamic worlds much more complex

References

- Poole &Mackworth, Artificial Intelligence: Foundations of Computational Agents, Chapter 6
- I. Bratko, *Prolog Programming for Artificial Intelligence*, 4th Edition, Chapter 17.