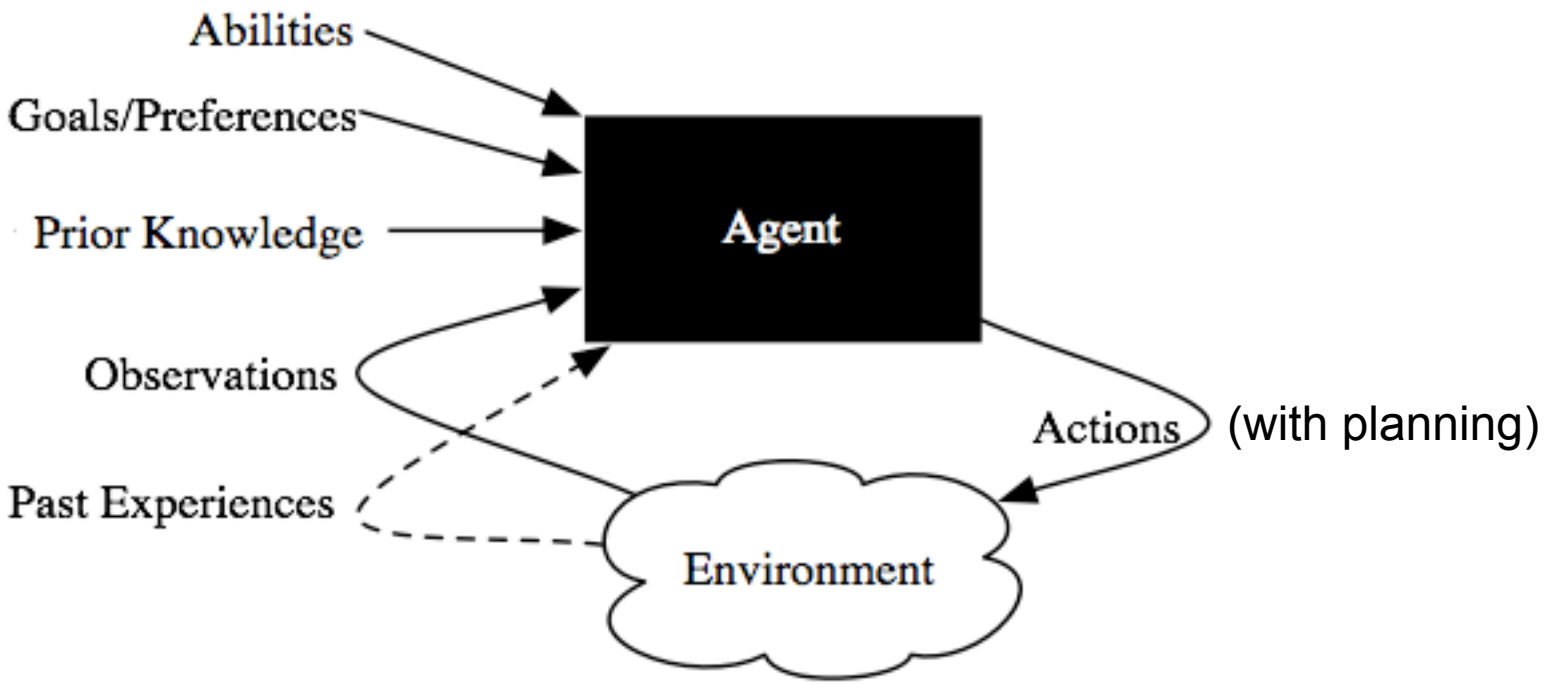
Planning

LECTURE 3 - part I

Reasoning About Action • STRIPS Planner
Forward planning • Regression Planning
Partial Order Planning • GraphPlan
Planning as Constraint Satisfaction

UNSW
SYDNEY

Agent acting in its environment



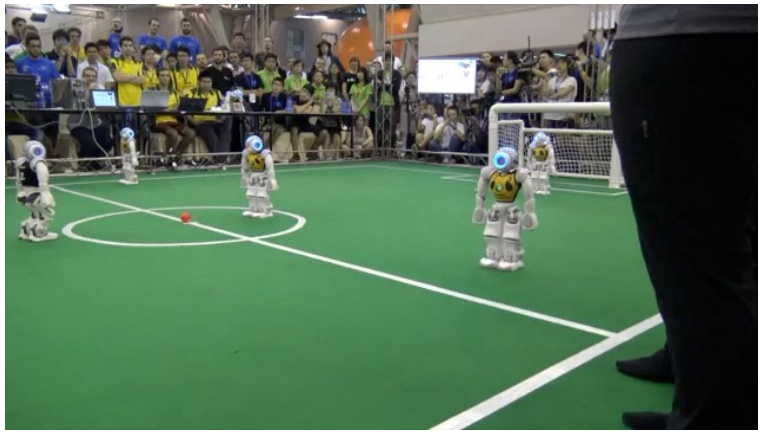
Planning

- 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.
- Assumptions:
  - World is deterministic.
  - No exogenous events outside of control of robot change state of 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 than a reactive agent because the knowledge that supports its decisions is represented explicitly and **can be modified.**
- The agent's behaviour can easily be changed.
- Doesn't work when assumptions are violated



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 soccer
  - Delivery robot
  - Self-driving car

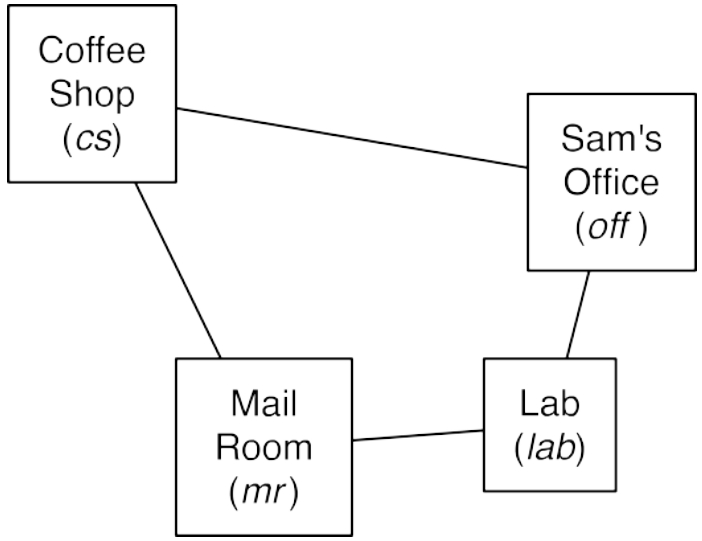
Representation

- How to represent a classical planning problem?
  - States, Actions, and Goals
- Can represent relation between states and actions
  - explicit state space representation
  - action-centric
  - feature-based

Actions

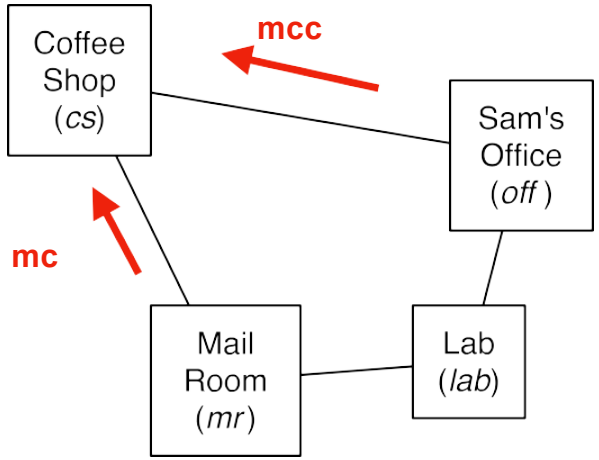
- A **deterministic action** is a partial function from states to states.
- The **preconditions** of an action specify when the action can be carried out.
- The **effect** of an action specifies the resulting state.

Delivery Robot Example



The robot, called Rob, can buy coffee at the coffee shop, pick upmail 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

Features to describe states

Actions: 顺时针的
mc – move clockwise
mcc – move counterclockwise 逆时针的
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,
  - coffee shop (cs), Sam's office (off), the mail room (mr) or 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 (puc) if Rob is at the coffee shop.
  - Rob can deliver coffee (dc) if Rob has coffee and is in the same location as Sam.
  - Rob can pick up mail (pum) if Rob is in the mail room.
  - Rob can deliver mail (dm) if Rob has mail and is in the same location as Sam.
- Assume that it is only possible for Rob to do one action at a time.

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)
  - efffects are conjunctions of relations

STRIPS Representation

- Each action has a:
  - •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.
  - •Often split into an ADD list (things that become true after action)
  - •and DELETE list (things that become false after action)

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

- 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 $\wedge$ ¬rhc

- Rules for "location is cs":

  RLoc′=cs $\leftarrow$ Rloc = off $\wedge$ Act=mcc

  RLoc′=cs $\leftarrow$ Rloc = mr $\wedge$ Act=mc

  RLoc′=cs $\leftarrow$ Rloc = cs $\wedge$ Act $\neq$ mcc $\wedge$ Act $\neq$ mc

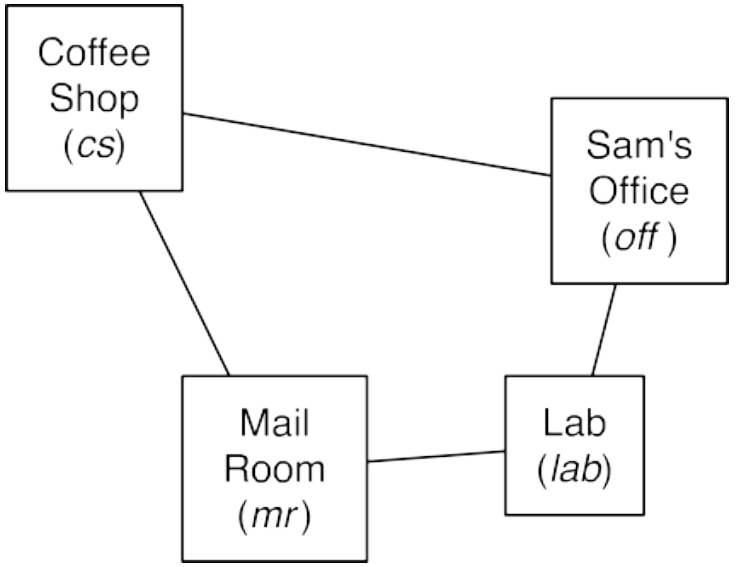
- Rules for "robot has coffee"

  rhc′ $\leftarrow$ Act=puc $\wedge$ ¬rhc

  rhc′ $\leftarrow$ rhc $\wedge$ Act $\neq$ dc

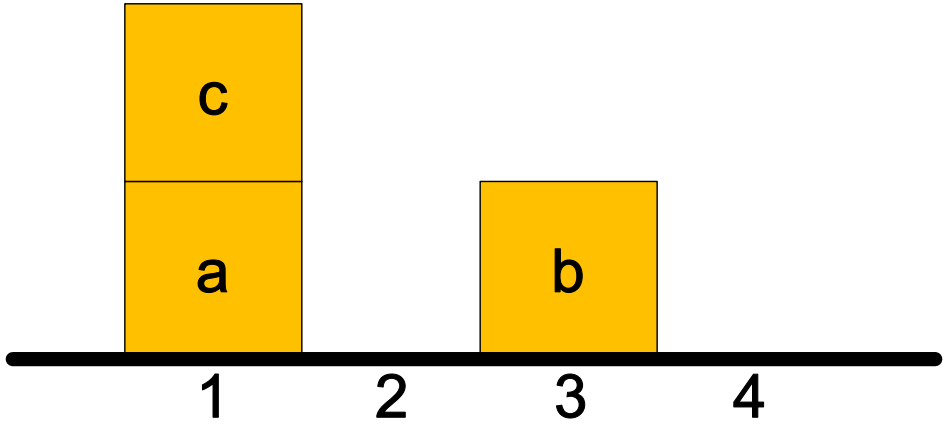
causal rules

frame rule



Relational State Representation

First-order representations are more flexible, e.g. states in blocks world

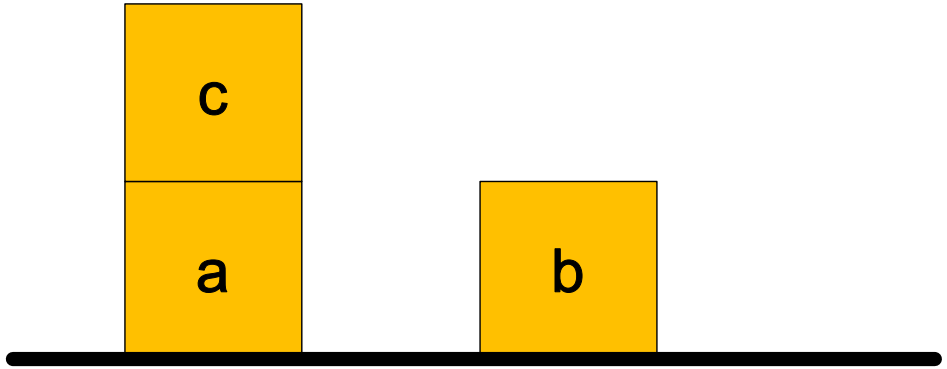


Can be represented by set of relations:

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:
  clear(a), on(a,1), clear(b)
- Action effects:
  on(a,b), clear(1), ~on(a,1), ~clear(b)



"add" (true after action)

"delete" (no longer true after action)
(~ = ¬)

STRIPS Action Schema

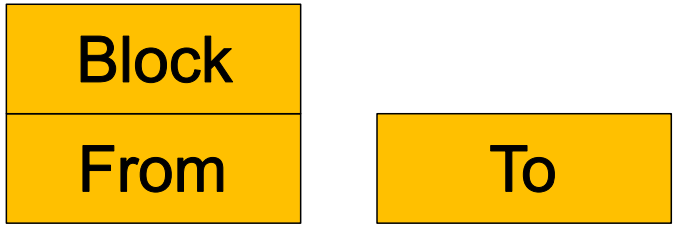
Action schema represents a set of actions using variables (variable names start with capital letter)

Action:
move(Block, From, To)

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

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

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



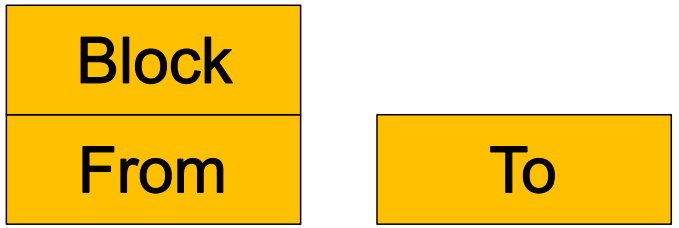
Better with Additional Constraints

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 ≠ Block, % Block cannot be moved to itself
object(From), % "From" is a block or a place
From ≠ To, % Move to new position
Block ≠ To

PDDL
Planning Domain Description Language

- Extension of STRIPS representation
- Invented for planning competitions to provide an implementation independent language for describing action schema and domain knowledge
- There are several variants to cover different planning domains
  - e.g. continuous domains, continuous actions, probabilities, etc.

Example

Init: $\text{Airport(MEL)} \wedge \text{Airport(SYD)} \wedge \text{Plane(P1)} \wedge \text{Plane(P2)} \wedge \text{Cargo(C1)} \wedge \text{Cargo(C1)} \wedge$
$\text{At(C1, SYD)} \wedge \text{At(C2, MEL)} \wedge \text{At(P1, SYD)} \wedge \text{At(P2, MEL)}$

Goal: $\text{At(C1, MEL)} \wedge \text{At(C2, SYD)}$

Action Load(c, p, a)
PRECOND: $\text{Cargo(c)} \wedge \text{Plane(p)} \wedge \text{Airport(a)} \wedge \text{At(c, a)} \wedge \text{At(p, a)}$
EFFECT: $\neg\, \text{At(c, a)} \wedge \text{In(c, p)}$

Action Unload(c, p, a)
PRECOND: $\text{Cargo(c)} \wedge \text{Plane(p)} \wedge \text{Airport(a)} \wedge \text{In(c, p)} \wedge \text{At(p, a)}$
EFFECT: $\text{At(c, a)} \wedge \neg\, \text{In(c, p)}$

Action Fly(p, from, to)
PRECOND: $\text{Plane(p)} \wedge \text{Airport(from)} \wedge \text{Airport(to)} \wedge \text{At(p, from)}$
EFFECT: $\neg\, \text{At(p, from)} \wedge \text{At(p, to)}$

$\text{Load}(C_1, P_1, \text{SYD})$

$\text{Fly}(P_1, \text{SYD}, \text{MEL})$

$\text{Unload}(C_1, P_1, \text{MEL})$

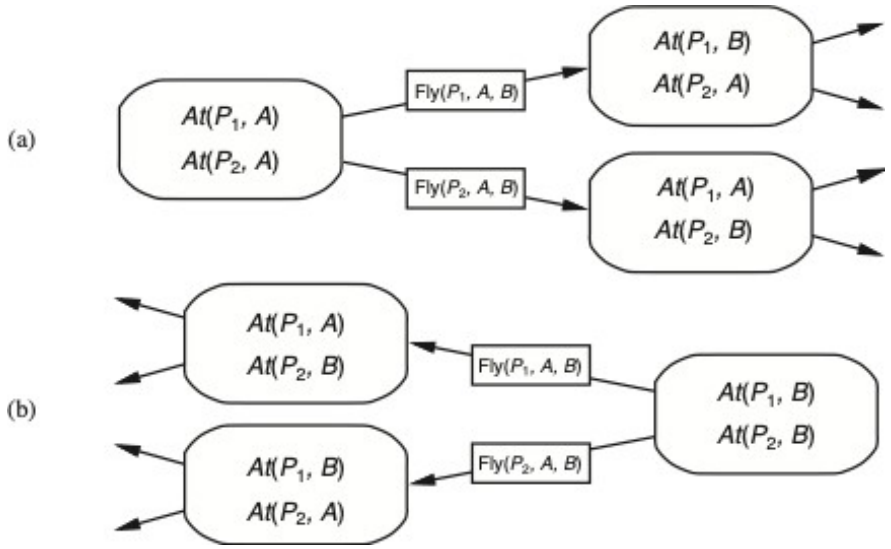
$\text{Load}(C_2, P_2, \text{MEL})$

$\text{Fly}(P_2, \text{MEL}, \text{SYD})$

$\text{Unload}(C_2, P_2, \text{SYD})$

Simple Planning Algorithms

Forward search and goal regression



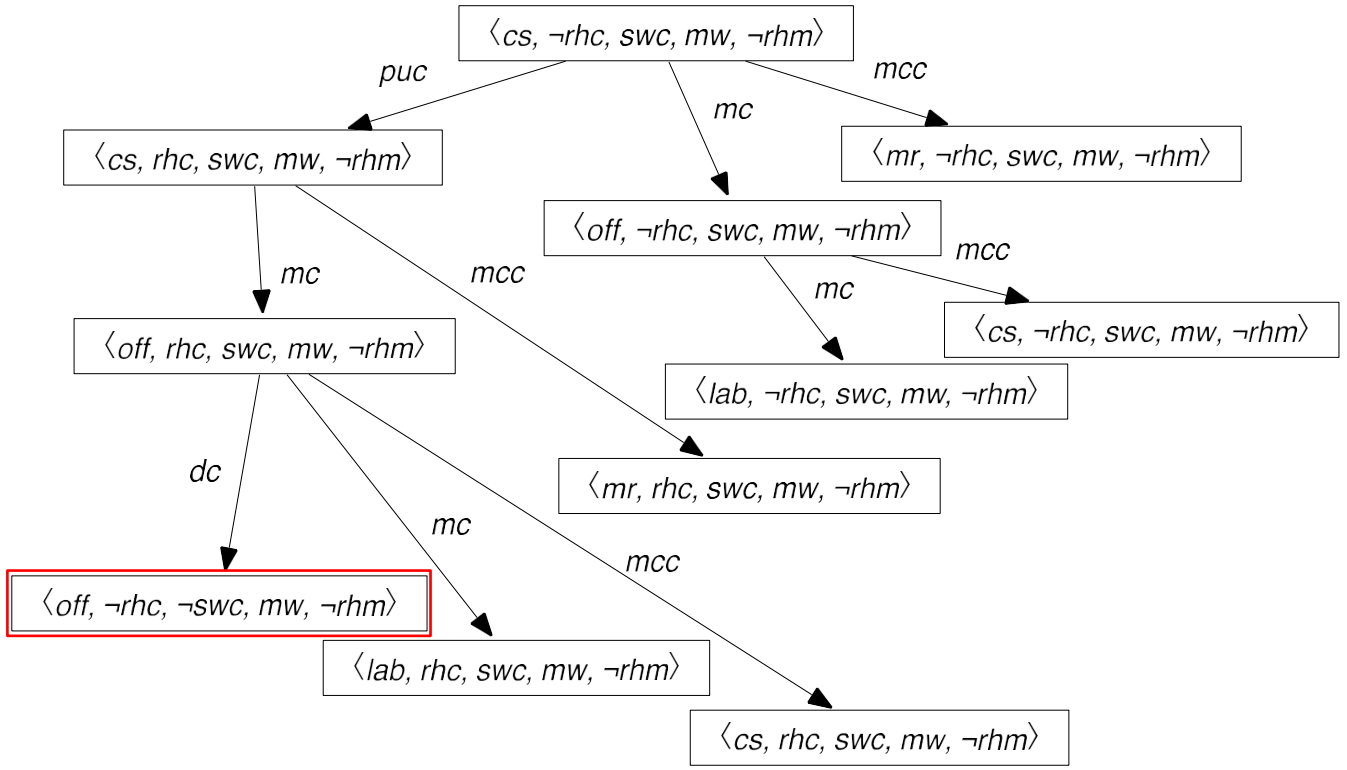
Forward Planning

- Nodes are states in the world
- Arcs correspond to actions that transform one state into another
- Start node is the initial state
- If *goal condition* is satisfied, search terminates successfully
- A path corresponds to a plan to achieve goal

Additional Details

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

Forward Search



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

Locations:
cs – coffee shop
off – office
lab – laboratory
mr – mail room

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

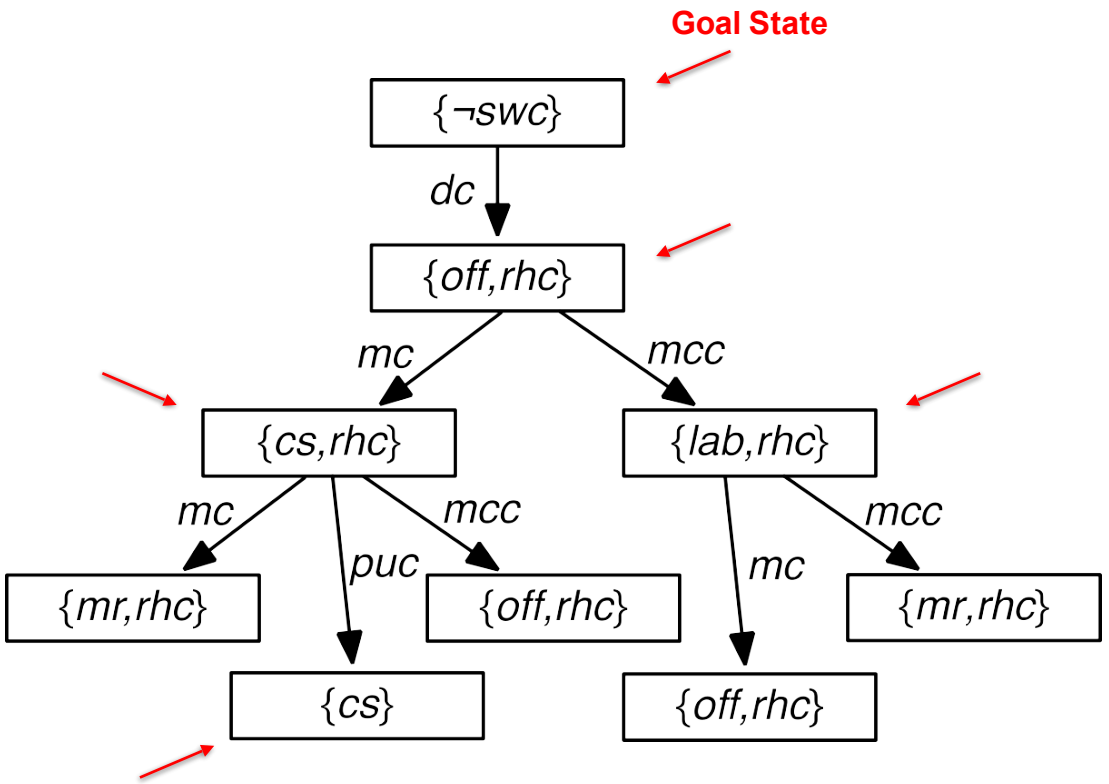
Initial:
SWC – Sam wants coffee

Goal:
¬*SWC* – Sam wants coffee

Regression Planning (Backward Search)

- Nodes are subgoals.
- Arcs correspond to actions. An arc from node *g* to *g′*, labelled 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.

Regression Planning



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

Locations:
cs – coffee shop
off – office
lab – laboratory
mr – mail room

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

Backward Regression

$$g' = (g - Add(a)) \cup Precond(a)$$

- g' is the regression from goal g over action a
- I.e. going backwards from g, we look for an action, a, that has preconditions and effects that satisfy g'

Sussman's Anomaly

- Goal: On(A, B) $\wedge$ On(B, C)
- Try achieving On(A, B) first

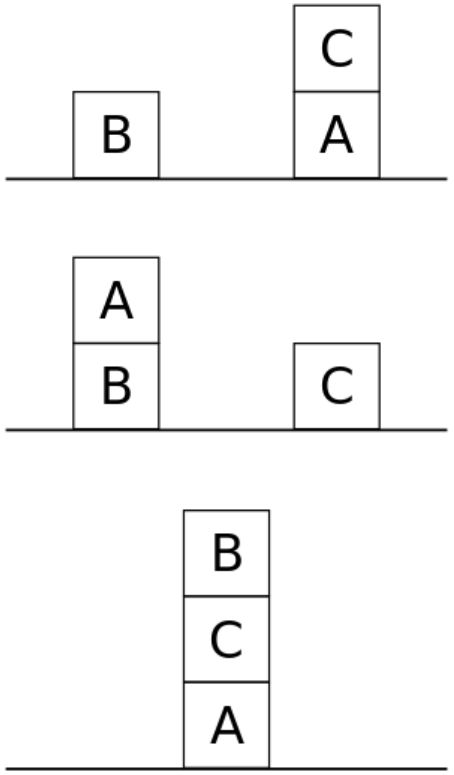
  [move(c,a,floor), move(a,floor,b),
  ***move(a,b,floor)**, move(b,floor,c)]*

- Trying On(B, C) first

  *[move(b,floor,c), **move(b,c,floor)**,*
  move(c,a,floor), move(a,floor,b)]

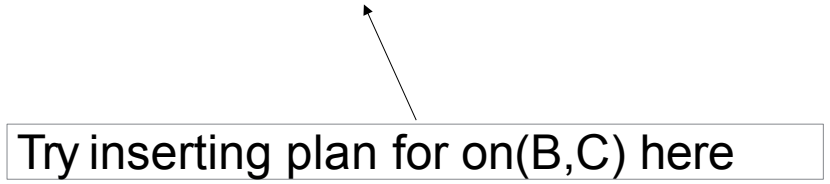
- Should be:

  [move(c,a,floor), move(b,floor,c), move(a,floor,b)]



Warplan

- *[move(c,a,floor), move(a,floor,b),* ***move(a,b,floor)****, ..]*
- *[move(c,a,floor), .., move(a,floor,b)]*



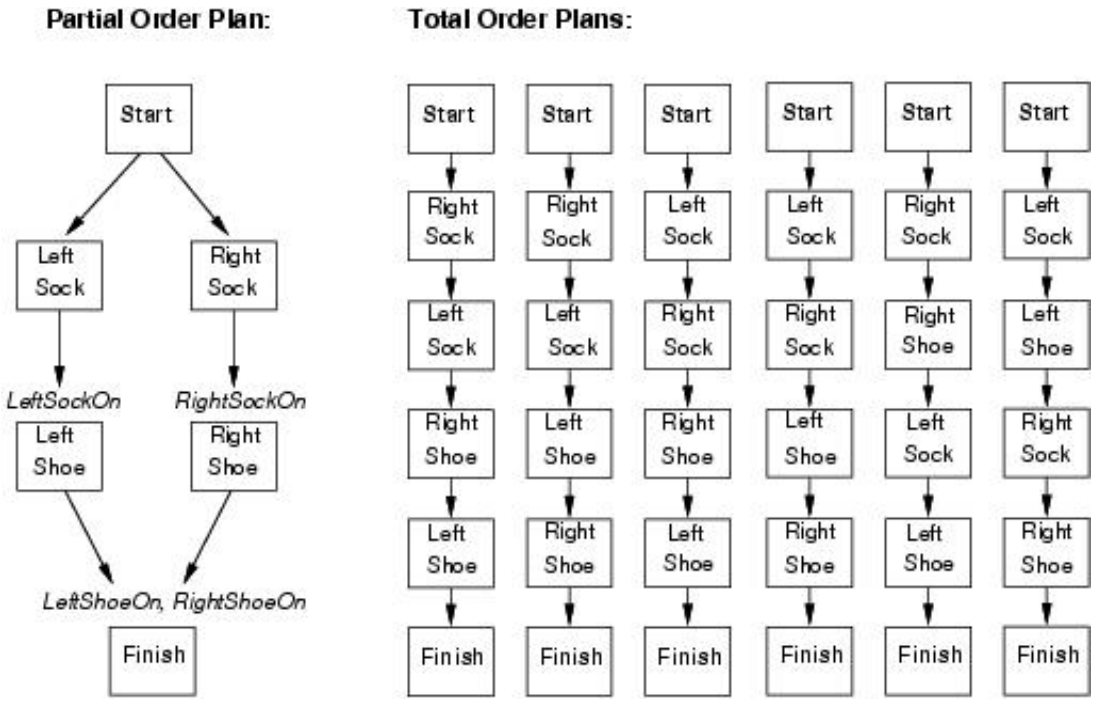
- check that goals before and after are preserved

WARPLAN

Warren, D. H. D. (1974). *Warplan: A system for generating plans.*
Memo No. 76, Department of Computational Logic, University of Edinburgh.

- WARPLAN tries to interleave actions by protecting goals.
  - Achieve on(A,B): [move(c,a,floor), move(a,floor,b)]
  - Protect on(A,B)
  - Now try on(B,C) by appending actions to end of plan
    - If it tries to undo a protected goal, move backwards through plan and try to slot new plan in.

Partially Ordered Plans



Partial-Order Planning

Init: Tire(Flat) ∧ Tire(Spare) ∧ At(Flat, Axle) ∧At(Spare, Boot) At

Goal: (Spare, Axle)

Action Remove(obj, loc)

PRECOND: At(obj, loc)

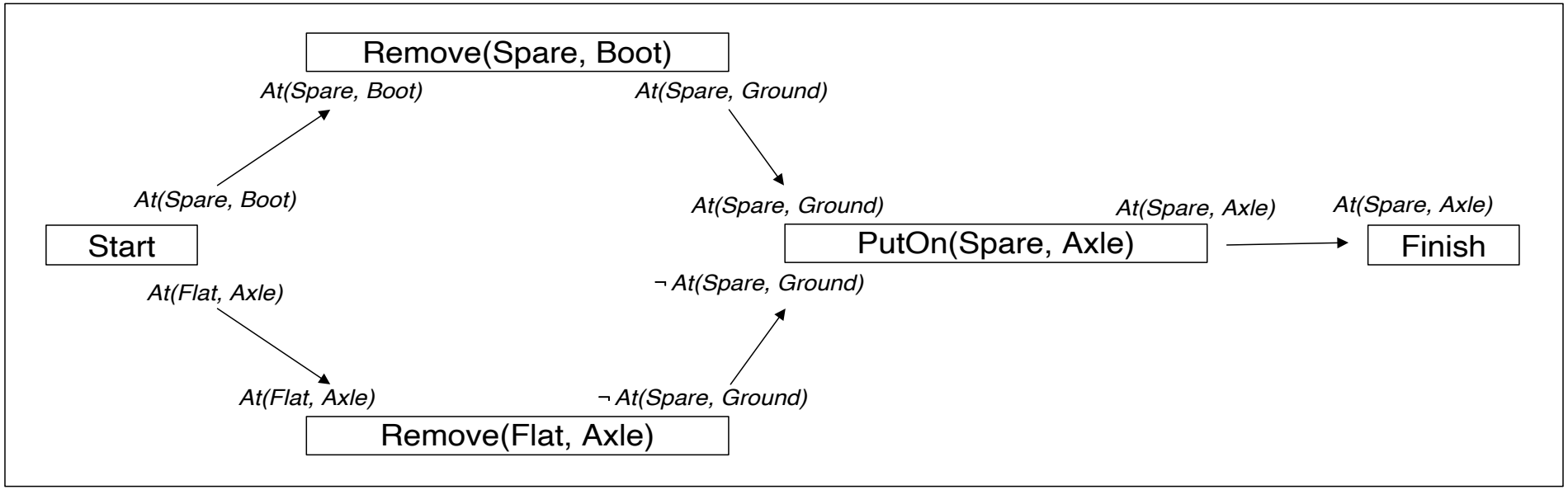
EFFECT: ¬ At(obj, loc) ∧ At(obj, Ground)

Action PutOn(t, Axle)

PRECOND: Tire(t) ∧ At(t, Ground) ∧ ¬ At(Flat,Axle)

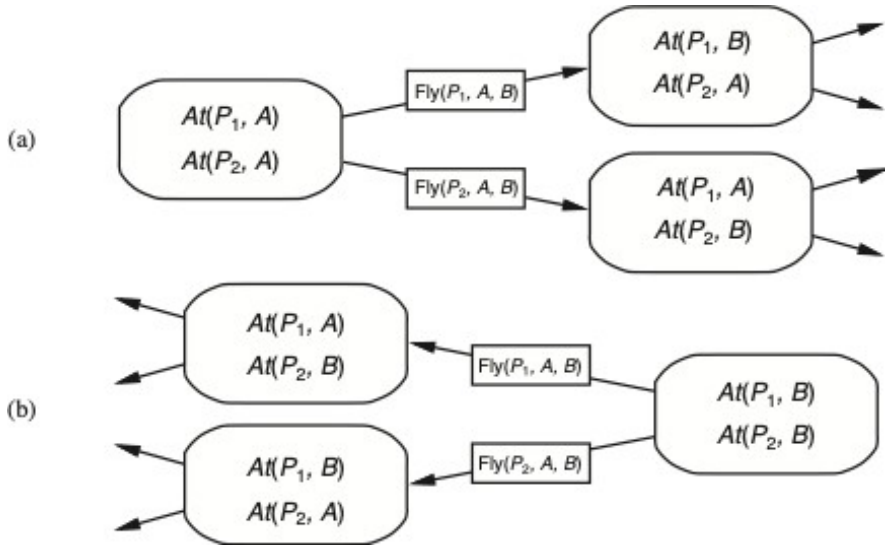
EFFECT: ¬ At(t, Ground) ∧ At(t,Axle)

Partial-Order Planning



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 Planning

- Forward planners are now among the best.
- Use heuristics to estimate costs
- Possible to use heuristic search, like A*, to reduce branching factor.

Planning Graphs

- Used constraint solving to achieve better heuristic estimates.
- Only for propositional problems
- Consists of a sequence of levels that correspond to time steps in the plan.
  - Level 0 is the initial state.
  - Each level consists of a set of literals and a set of actions.
    - Literals = all those that could be true at that time step, depending upon the actions executed at the preceding time step.
    - Actions = all those actions that could have their preconditions satisfied at that time step, depending on which of the literals actually hold.

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

Example

Init: Have (Cake)

Goal: Have(Cake) ∧ Eaten(Cake)

Action: Eat (Cake)

PRECOND: Have(Cake)

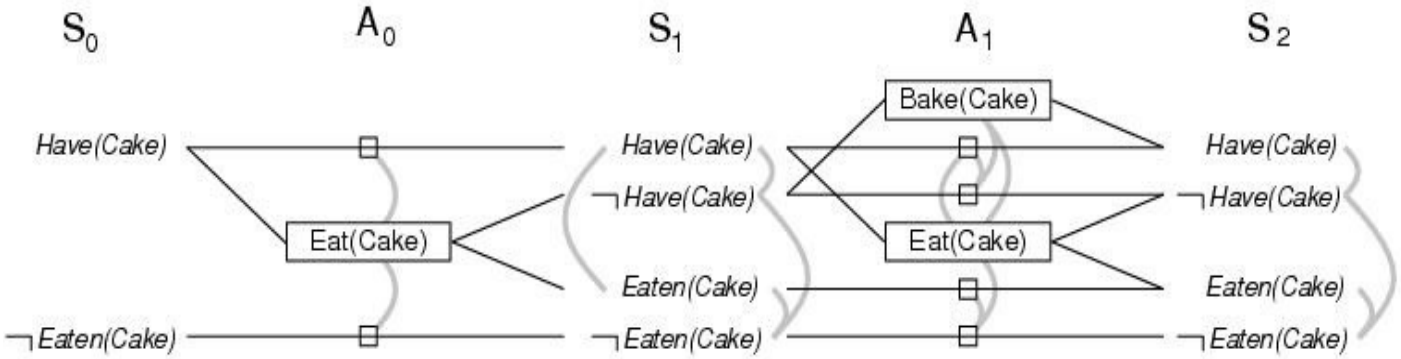
EFFECT: ¬ Have(Cake) ∧ Eaten(Cake)

Action: Bake (Cake)

PRECOND: ¬ Have(Cake)

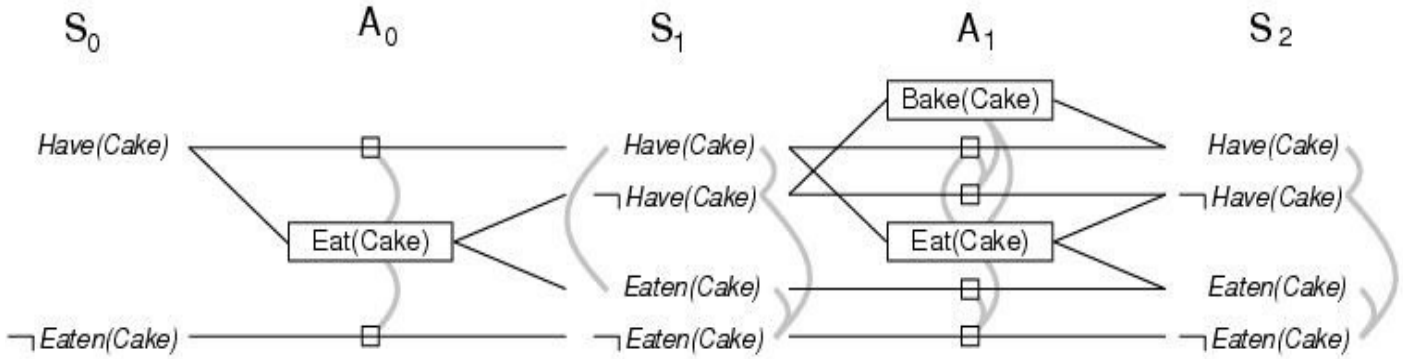
EFFECT: Have(Cake)

Cake Example



- Start at level S0 and determine action level A0 and next level S1.
- Inaction is represented by persistence actions.
  - A0 – all actions whose preconditions are satisfied in the previous level.
  - Connect precondition and effect of actions S0 → S1
  - Inaction is represented by persistence actions
- Level A0 contains the actions that could occur
  - Conflicts between actions are represented by mutex links

Cake Example



- Level S1 contains all literals that could result from picking any subset of actions in A0
  - Conflicts between literals that cannot occur together are represented by mutex links.
  - S1 defines multiple states and the mutex links are the constraints that define this set of states.
- Continue until two consecutive levels are identical: levelled off

Planning Graphs and Heuristic Estimation

- Planning Graphs provide information about the problem
  - A literal that does not appear in the final level of the graph cannot be achieved by any plan.
    - Useful for backward search (cost = inf).
  - Level of appearance can be used as cost estimate of achieving any goal literals = level cost.
  - Cost of a conjunction of goals? Max-level, sum-level and set-level heuristics.

Example: Spare tire problem

Init: At(Flat, Axle) ∧ At(Spare,Trunk)

Goal: At(Spare,Axle)

Action: Remove(Spare,Trunk)

PRECOND: At(Spare,Trunk)

EFFECT: ¬At(Spare,Trunk) ∧ At(Spare,Ground))

Action: Remove(Flat,Axle)

PRECOND: At(Flat,Axle)

EFFECT: ¬At(Flat,Axle) ∧ At(Flat,Ground))

Action: PutOn(Spare,Axle)

PRECOND: At(Spare,Ground) ∧¬At(Flat,Axle)

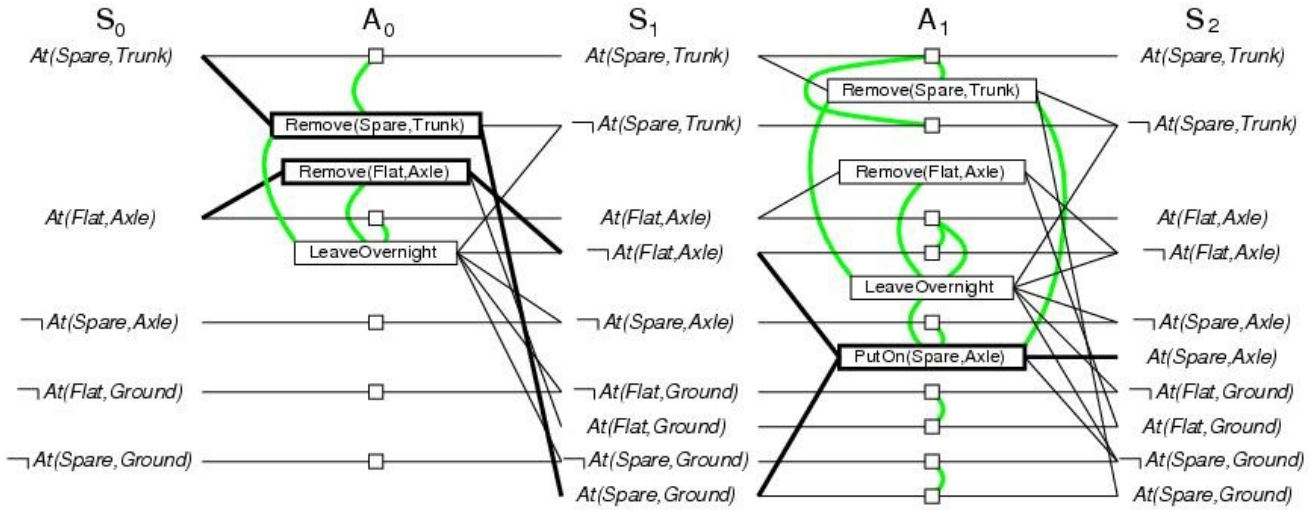
EFFECT: At(Spare,Axle) ∧ ¬At(Spare,Ground))

Action: LeaveOvernight

PRECOND:

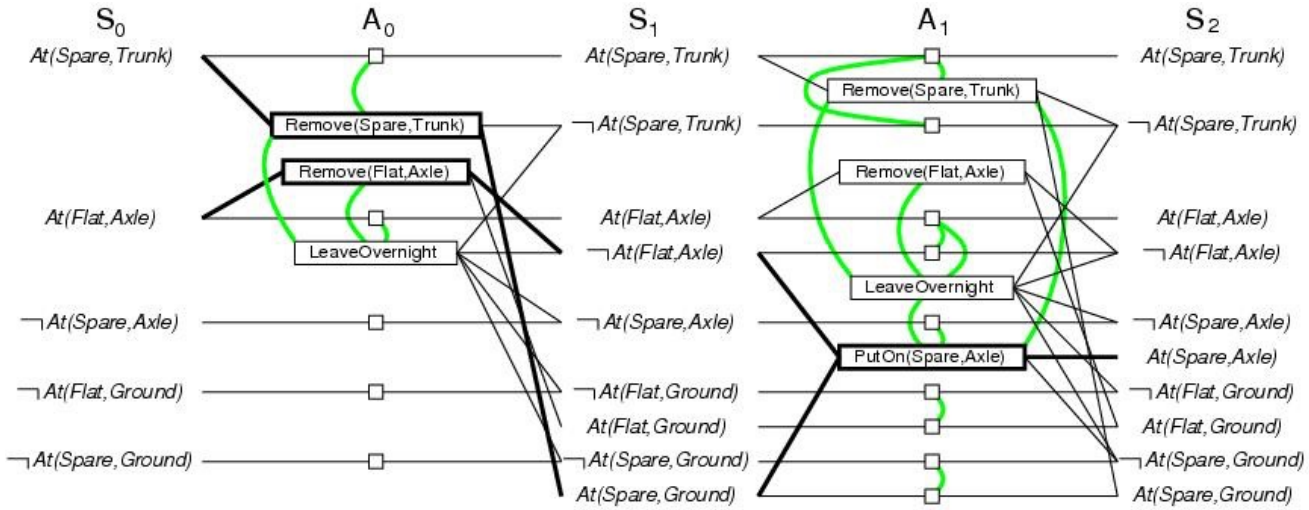
EFFECT: ¬ At(Spare,Ground) ∧ ¬ At(Spare,Axle) ∧ ¬ At(Spare,trunk) ∧ ¬ At(Flat,Ground) ∧ ¬ At(Flat,Axle))

GRAPHPLAN Example



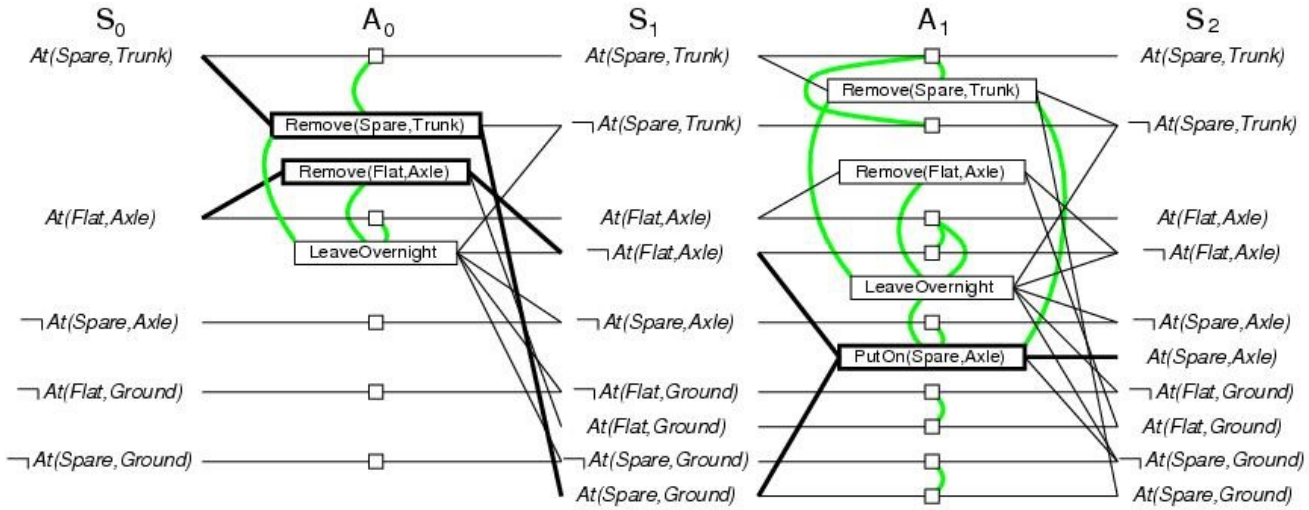
- *Initially the plan consist of literals from the initial state and literals from the closed world assumption (S0).*
- *Add actions whose preconditions are satisfied by EXPAND-GRAPH (A0)*
- *Also add persistence actions and mutex relations.*
- *Add the effects at level S1*
- *Repeat until goal is in level Si*

GRAPHPLAN example



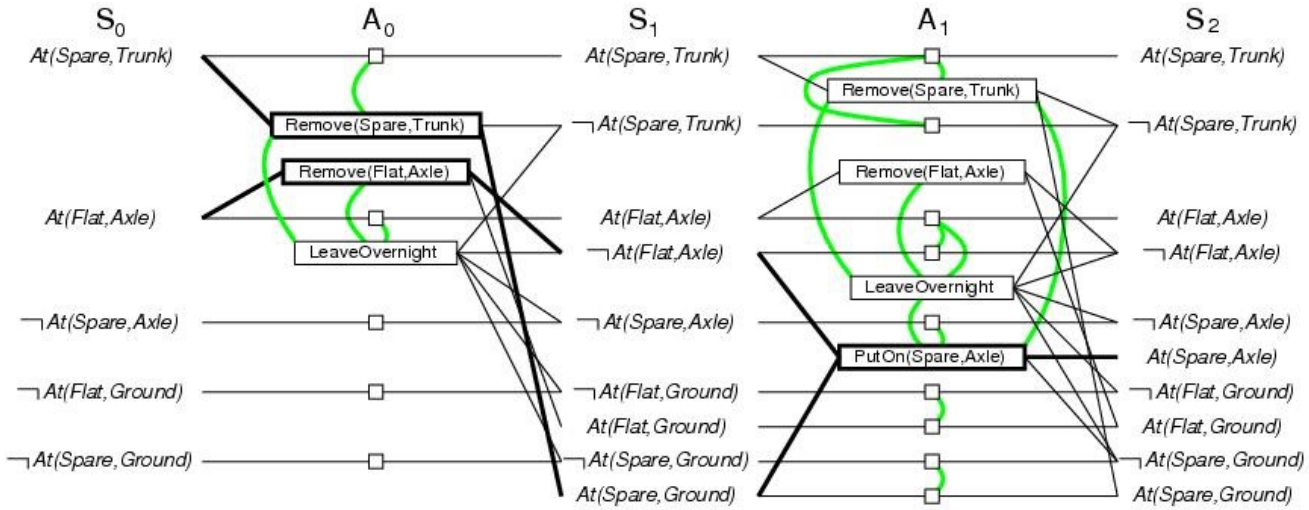
- *EXPAND-GRAPH looks for mutex relations*
  - *Inconsistent effects*
    - *E.g. Remove(Spare, Trunk) and LeaveOverNight due to At(Spare,Ground) and* **not** *At(Spare, Ground)*
  - *Interference*
    - *E.g. Remove(Flat, Axle) and LeaveOverNight At(Flat, Axle) as PRECOND and* **not** *At(Flat,Axle) as EFFECT*
  - *Competing needs*
    - *E.g. PutOn(Spare,Axle) and Remove(Flat, Axle) due to At(Flat, Axle) and* **not** *At(Flat, Axle)*
  - *Inconsistent support*
    - *E.g. in S2, At(Spare,Axle) and At(Flat,Axle)*

GRAPHPLAN Example



- *In S2, the goal literals exist and are not mutex with any other*
  - *Solution might exist and EXTRACT-SOLUTION will try to find it*
- *EXTRACT-SOLUTION can use Boolean CSP to solve the problem or a search process:*
  - *Initial state = last level of PG and goal goals of planning problem*
  - *Actions = select any set of non-conflicting actions that cover the goals in the state*
  - *Goal = reach level S0 such that all goals are satisfied*
  - *Cost = 1 for each action.*

GRAPHPLAN Example



- *Termination? YES*
- *PG are monotonically increasing or decreasing:*
  - *Literals increase monotonically*
  - *Actions increase monotonically*
  - *Mutexes decrease monotonically*
- *Because of these properties and because there is a finite number of actions and literals, every PG will eventually level off!*

Extracting the Plan

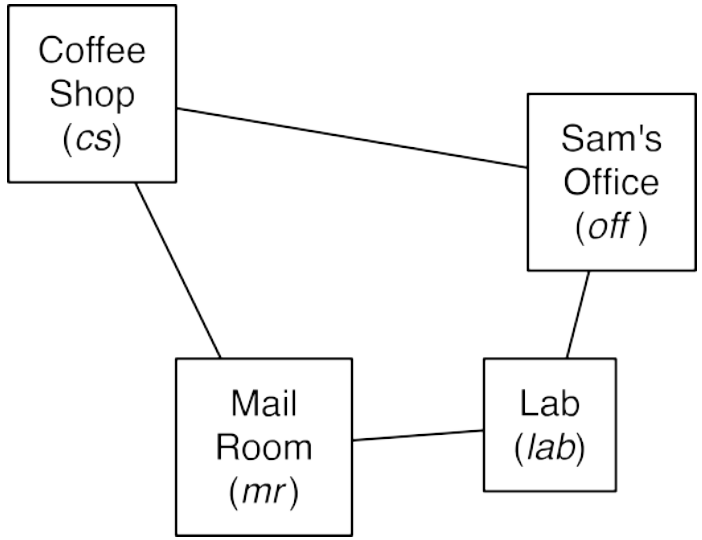
- Heuristic forward search planners use A* to find path from start to goal
- Cost is based on level in graph

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

Planning as Constraint Satisfaction



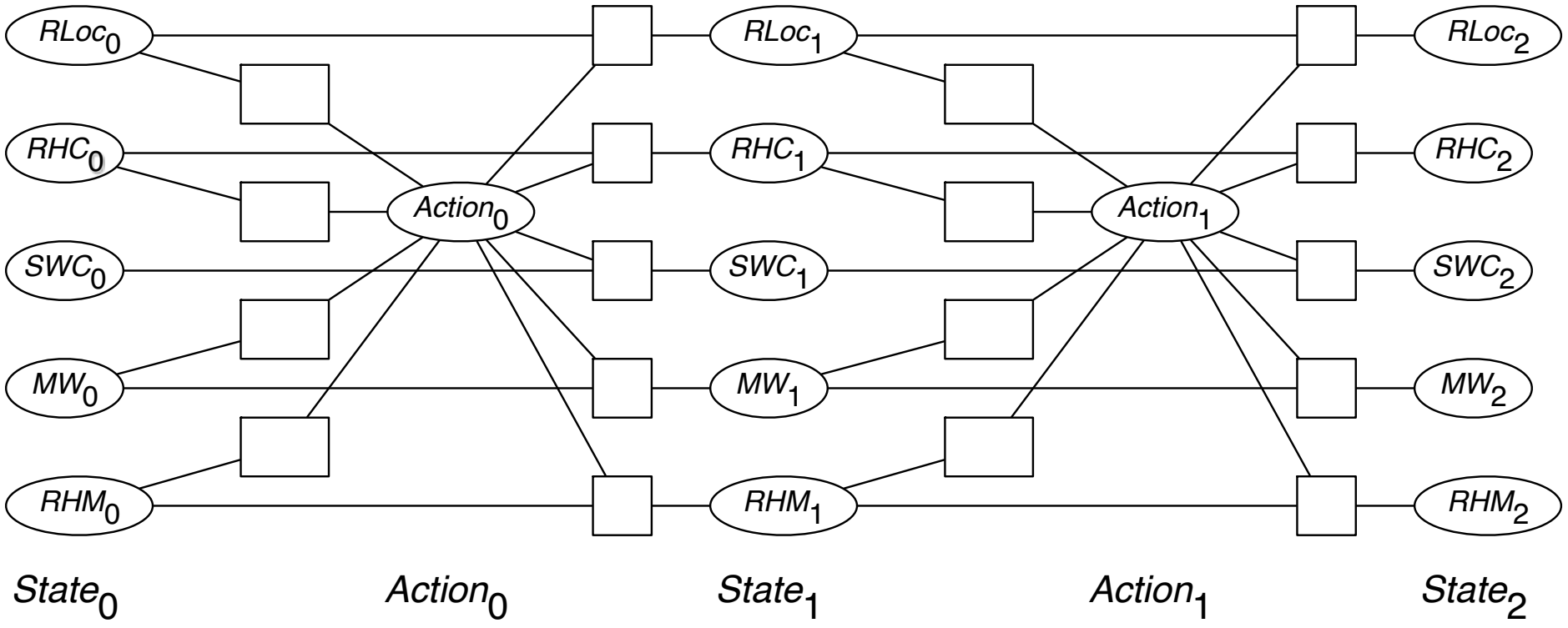
Features:
RLoc – Rob's location
RHC – Rob has coffee
SWC – Sam wants coffee
MW – Mail is waiting
RHM – Rob has mail

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

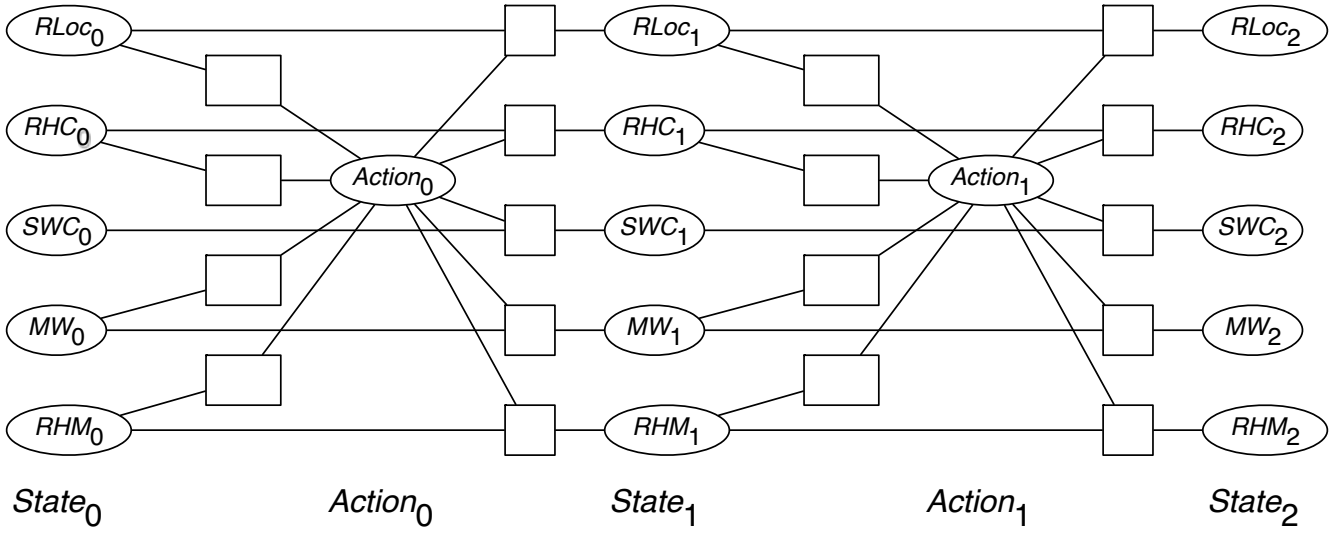
Planning as Constraint Satisfaction



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.

- 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.
- 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