Planning

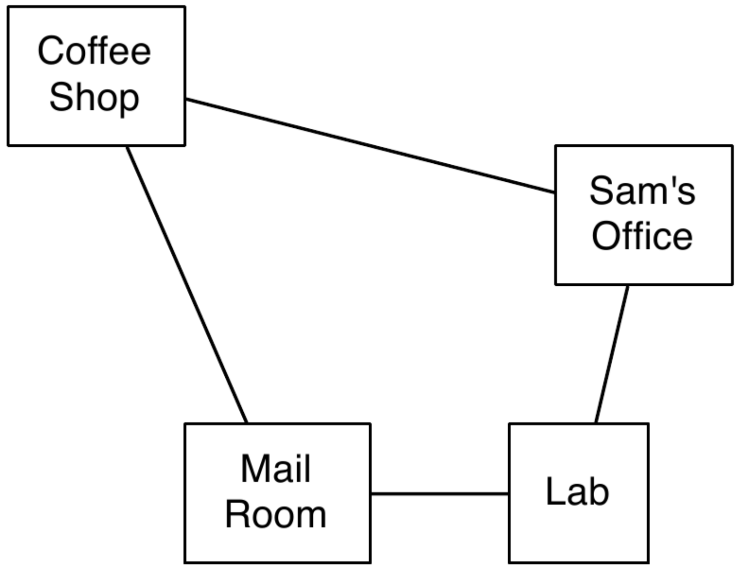
Definition and Assumptions

- *Planning* is finding actions to achieve goals.
- Initial assumptions:
  - The world is deterministic.
  - There are no events outside of the control of the agents 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 features of states that need to be achieved or maintained.

Actions

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

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
nm: no move
puc: pickup coffee
dc: deliver coffee
pum: pickup mail
dm: deliver mail

Explicit State Space Representation

State	Action	Resulting State
$lab, \overline{rhc}, swc, \overline{mw}, rhm$	mc	$mr, \overline{rhc}, swc, \overline{mw}, rhm$
$lab, \overline{rhc}, swc, \overline{mw}, rhm$	mcc	$off, \overline{rhc}, swc, \overline{mw}, rhm$
$off, \overline{rhc}, swc, \overline{mw}, rhm$	dm	$off, \overline{rhc}, swc, \overline{mw}, \overline{rhm}$
$off, \overline{rhc}, swc, \overline{mw}, rhm$	mcc	$cs, \overline{rhc}, swc, \overline{mw}, rhm$
$off, \overline{rhc}, swc, \overline{mw}, rhm$	mc	$lab, \overline{rhc}, swc, \overline{mw}, rhm$
...	...	...

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
nm: no move
puc: pickup coffee
dc: deliver coffee
pum: pickup mail
dm: deliver mail

Feature-Based Representation of Actions

For each action:

- A *precondition* specifies when the action can be carried out.

For each feature:

- *Causal rules* specify when the feature gets a new value.
- *Frame rules* specify when the feature keeps its value.

Example Feature-Based Representation

Precondition of pick-up coffee (puc):

$$Act = puc \rightarrow RLoc = cs \wedge \neg rhc$$

Rules for next location = coffee shop ($RLoc' = cs$):

$$RLoc = off \wedge Act = mcc \rightarrow RLoc' = cs$$
$$RLoc = mr \wedge Act = mc \rightarrow RLoc' = cs$$
$$RLoc = cs \wedge Act \neq mcc \wedge Act \neq mc \rightarrow RLoc' = cs$$

Rules for “robot has coffee” (rhc)

$$rhc \wedge Act \neq dc \rightarrow rhc'$$
$$Act = puc \rightarrow rhc'$$

STRIPS Representation

For each action:

- A *precondition* specifies when the action can be carried out.
- An *effect* assigns values to features that are changed by this action.

Action: Pick-up coffee (puc):

- Precondition: $RLoc = cs \land \neg rhc$
- Effect: rhc

Action: Deliver coffee (dc):

- Precondition: $off \land rhc$
- Effect: $\neg rhc \land \neg swc$

Planning Problem

Given:

- A description of the effects and preconditions of the actions
- A description of the initial state
- A goal to achieve

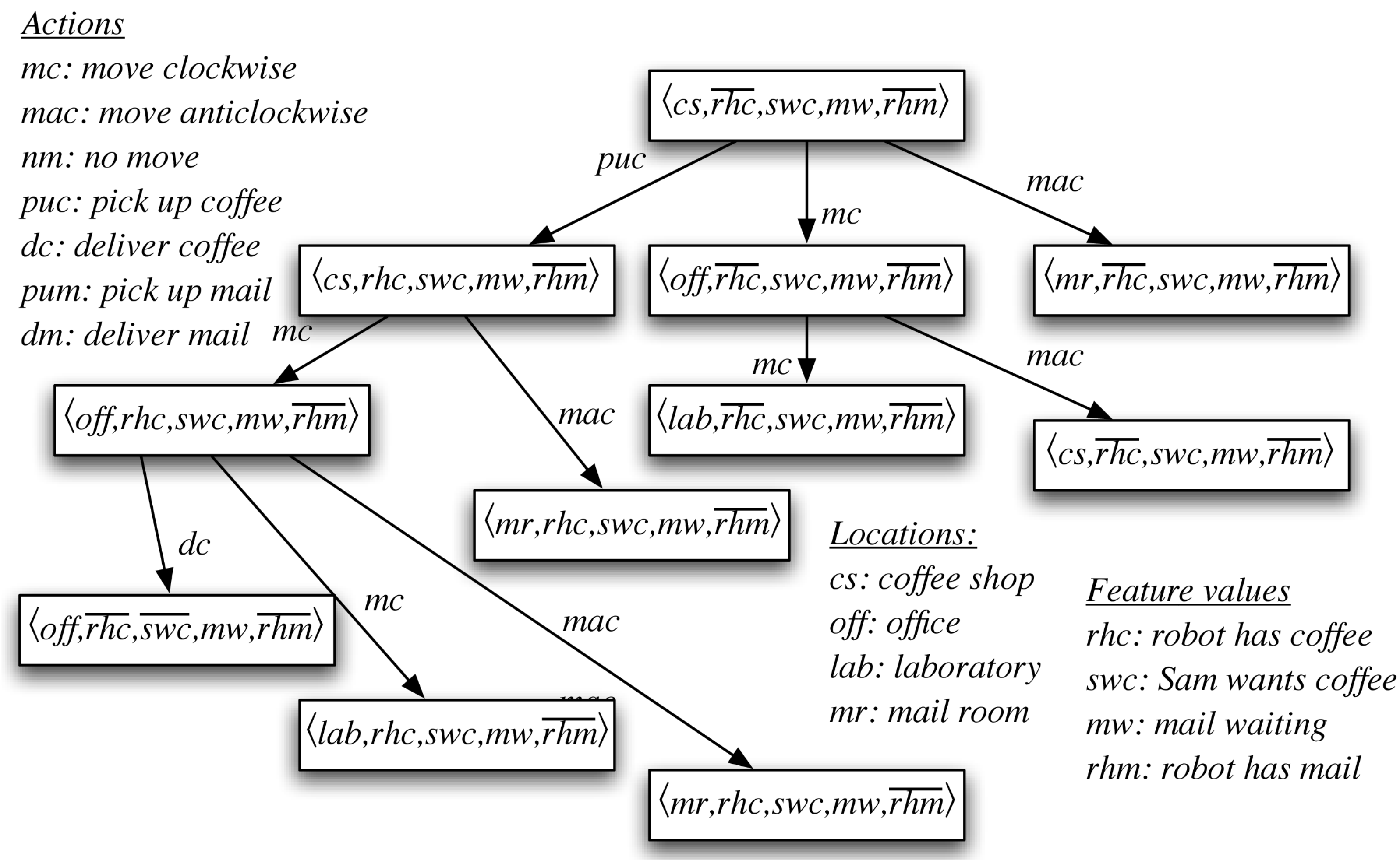
find a sequence of actions that is possible and will result in a state satisfying the goal.

Forward Planning

Idea: search in the state-space graph.

- The nodes represent the states
- The arcs correspond to the actions: The edges from a state s represent all of the actions that are legal in state s.
- A plan is a path from the state representing the initial state to a state that satisfies the goal.

Forward Planning Example



Forward Planning Comments

- The search graph can be constructed on demand: you only construct reachable states.
- Forward search can use knowledge specified as:
  - a heuristic function that estimates the number of steps to the goal
  - domain-specific pruning of neighbors:
    - don't pick-up coffee unless Sam wants coffee
    - unless the goal involves time constraints, don't do the "no move" action.

Regression Planning

Idea: search backwards from the goal description: nodes correspond to goals and subgoals, and arcs to actions that achieve goals.

- Nodes are partial assignments of values to features.
- Edges correspond to actions that can achieve one of the assignments.
- The edge points to a node that includes the preconditions of the action.
- The initial node is the goal to be achieved.
- Search succeeds if a node is true of the initial state.

Goals, Subgoals and Edges

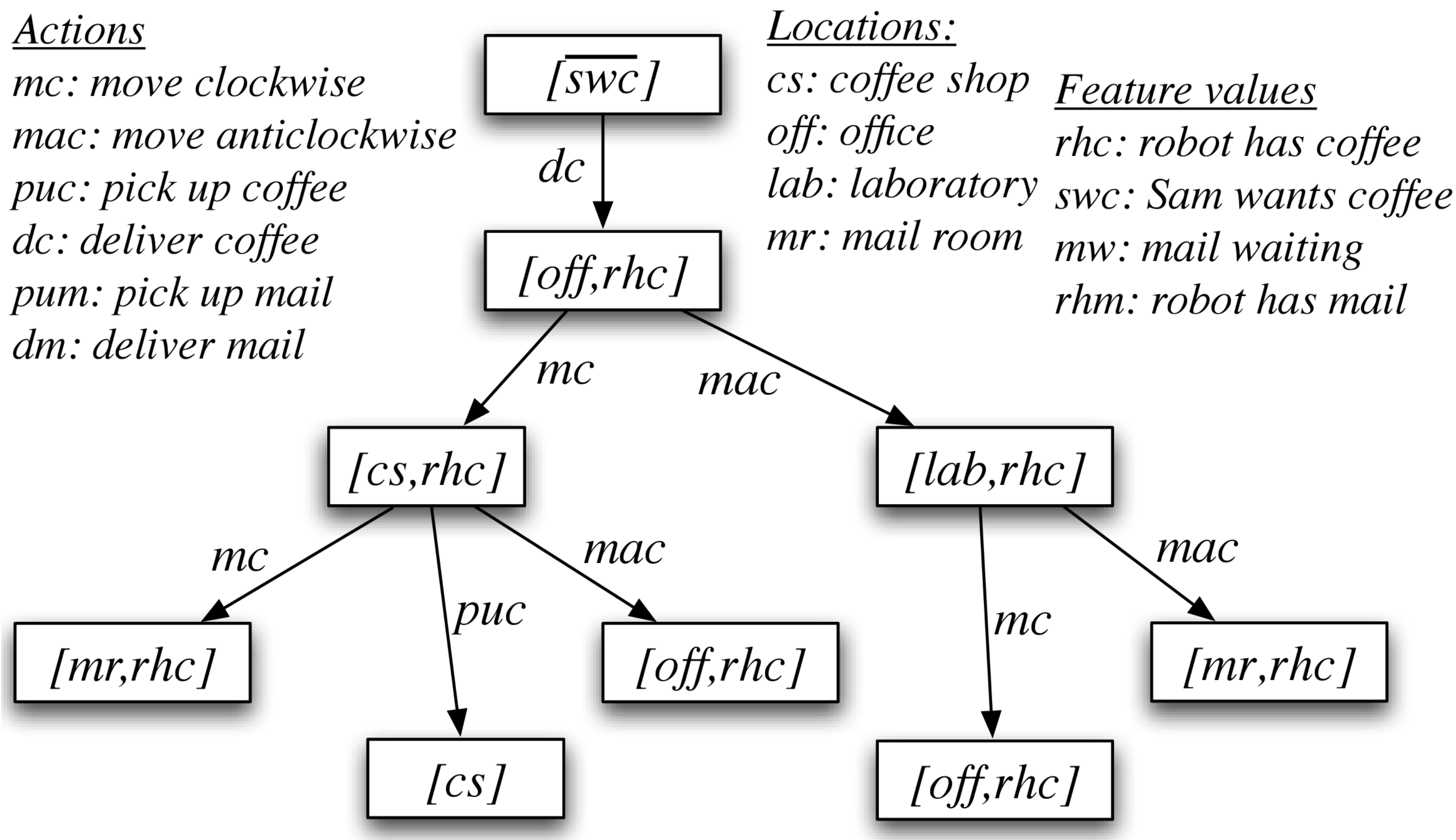
- A node g represents goals (or subgoals) to be achieved: represented as a value assignment to one or more features:

$$X_i = v_i, X_j = v_j, \ldots$$

- An action from g includes part of g as an effect, with no effect that contradicts g.
- The edge goes to a node g' that must contain:
  - The preconditions of the action
  - All elements of g not in the action's effect

  g' must not have contradictions.

Regression Example



Regression Planning Comments

- You can define a heuristic function that estimates how difficult it is to achieve a node from the initial state.
- You can use domain-specific knowledge to remove impossible goals.
- Whether forward or regression is more efficient depends on the branching factor and how good the heuristics are.
- Forward planning is unconstrained by the goal (except as a source of heuristics).
- Regression planning is unconstrained by the initial state (except as a source of heuristics)

Planning as Constraint Satisfaction Problems

- Idea: Create a CSP for a limited-length plan.
- If length k fails, increment k and try again.
- Algorithm:
  - Choose a plan length k (also called the *horizon*).
  - Create a variable for each feature and each time from 0 to k.
  - Create a variable for each action for each time in the range 0 to $k-1$.
  - Add constaints between features and actions, and solve.
- Very effective with a specialized algorithm.

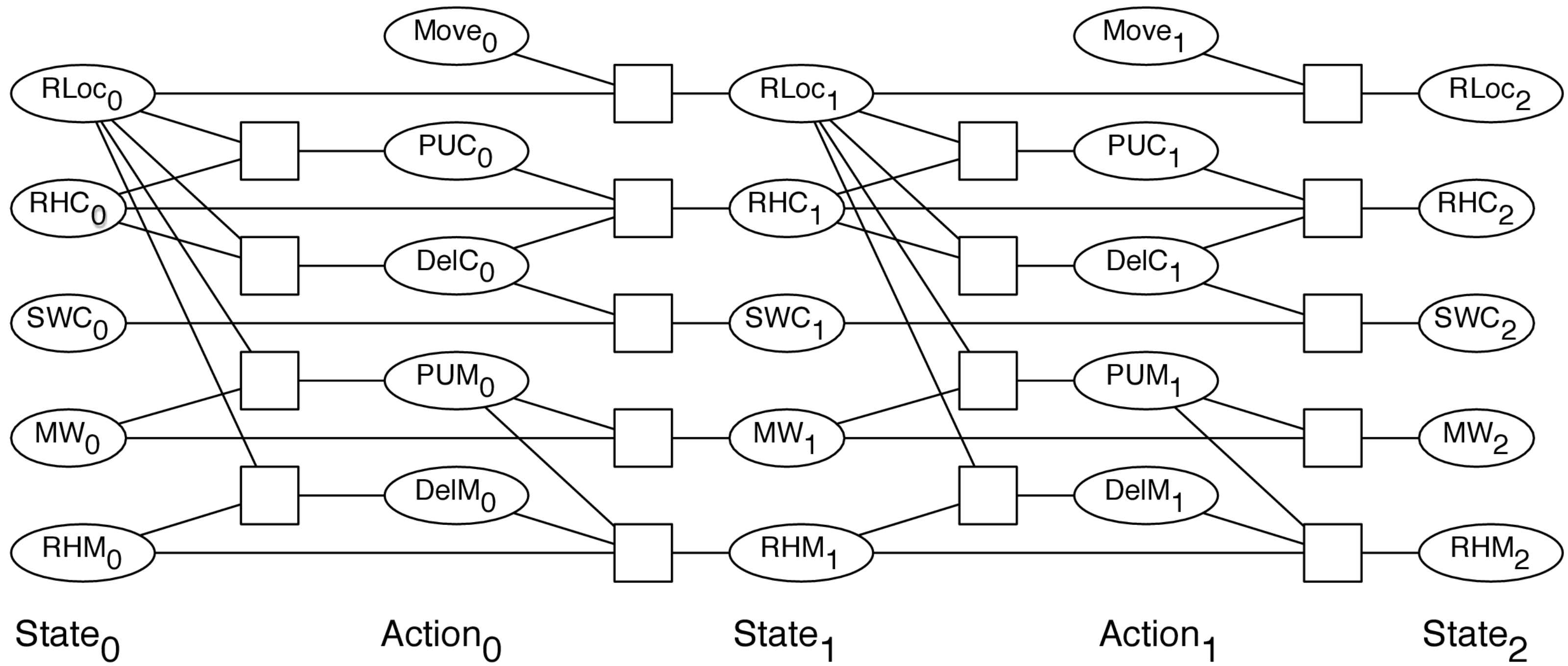
Action Variables

- PUC: Boolean var, robot picks up coffee.
- $DelC$: Boolean var, robot delivers coffee.
- PUM: Boolean var, robot picks up mail.
- $DelM$: Boolean variable, robot delivers mail.
- $Move$: variable with domain $\{mc, mcc, nm\}$ specifies whether the robot moves clockwise, counterclockwise or doesn't move

Constraints

- *State constraints* between variables at the same time step.
- *Precondition constraints* between state vars at time t and action vars at time t.
- *Effect constraints* between state vars at time t, action vars at time t, and state vars at time $t+1$.
- *Action constraints* specify which actions cannot co-occur (also called *mutex constraints*).
- *Initial state constraints* on the state at time 0.
- *Goal constraints* specify that goals are satisfied at time k.

CSP for Delivery Robot



$RLoc_i$ — 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

Example Constraints

Precondition Constraint

RHC_i	$RLoc_i$	PUC_i
false	cs	true
any	any	false

Effect Constraint

RHC_i	DC_i	PUC_i	RHC_{i+1}
true	true	false	false
true	false	false	true
false	false	true	true
false	false	false	false

Action Constraint

$Move_i$	PUC_i
nm	true
any	false

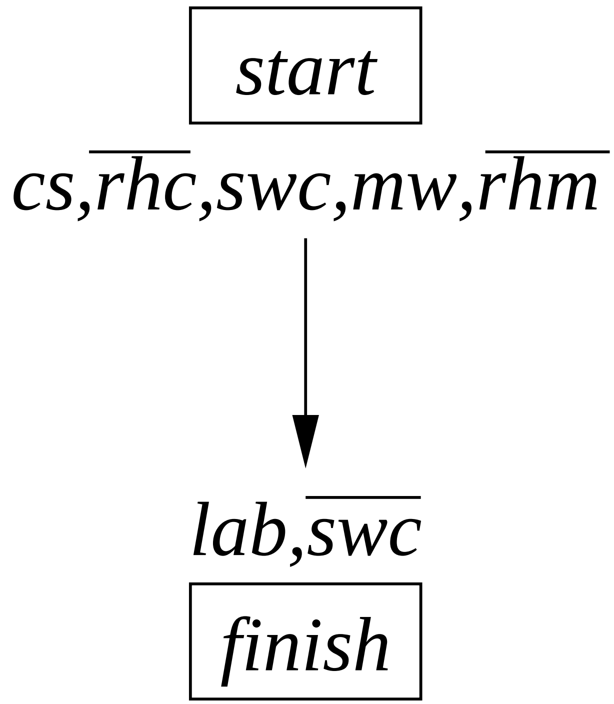
Partial Order Planning

- Forward, regression and CSP planners commit to unnecessary action orderings.
- Idea: Maintain a partial ordering between actions and only commit to an ordering between actions when forced.
- A *partial-order plan* is a partial ordering of actions ($act_0 < act_1$ represents act_0 before act_1). The problem is solved when every total ordering is a solution.
- Algorithm Idea: Start with an unfinished plan and search over ways to fix it.

Partial Plan Search

Procedure *Partial-Plan-Search*(s, g, A)
  Inputs: s, g, A: initial state, goal, actions
  start $\leftarrow$ pseudo-action with s as effect
  finish $\leftarrow$ pseudo-action with g as precondition
  insert plan *start* $<$ *finish* into *Frontier*
  while *Frontier* is not empty
    $p \leftarrow$ remove a plan from *Frontier*
    if p has no flaws then return p
    select a flaw w in p
    for each fix x for w in p
      $p' \leftarrow$ copy of p including x
      insert p' into *Frontier*
  return null

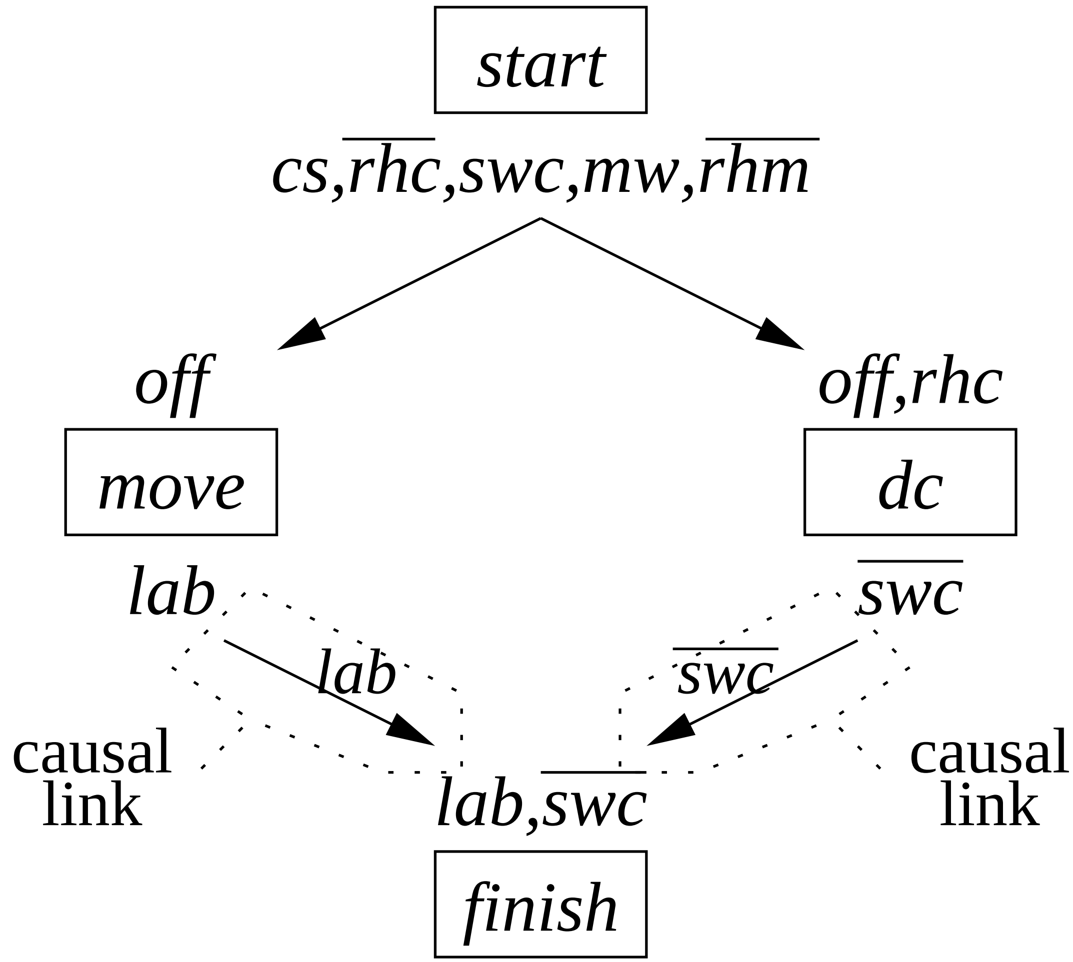
Initial Plan



flaw: open precondition: lab not achieved
fix: add $move$ from off to lab

flaw: open precondition: $\overline{swc}$ not achieved
fix: add dc action

Intermediate Plan



Flaws in Intermediate Plan

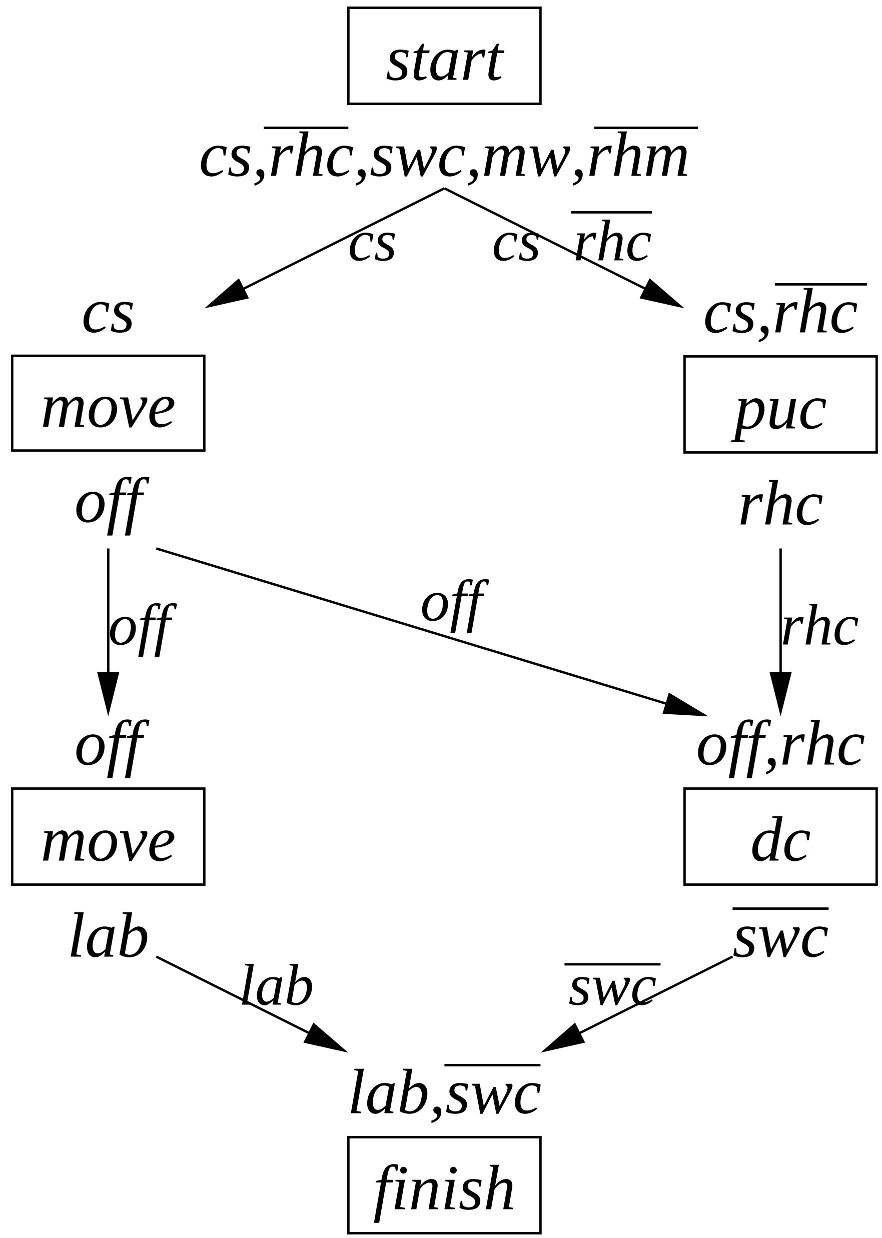
flaw: open precondition: off of $move$ not achieved
fix: add $move$ from cs to off

flaw: open precondition: off of dc not achieved
fix: use same $move$ from cs to off

flaw: open precondition: rhc of dc not achieved
fix: add puc action

flaw: open preconditions of puc and new $move$
fix: use effects of $start$ (use initial state)

Almost Done Plan



Flaws in Almost Done Plan

flaw: conflict: the first $move$ conflicts with cs staying true between $start$ and puc

fix: order first $move$ after puc

flaw: conflict: the second $move$ conflicts with off staying true between first $move$ and dc.

fix: order second $move$ after dc

Comments

- The above example doesn't show the search over all the fixes that don't work.
- Works well if plans for different subgoals do not interact much.
- Compared to forward/regression planning, adds search levels for reusing actions and resolving conflicts.
- In practice, CSP planners are more efficient than partial order/forward/regression planning.