Planning Under Certainty

Planning Under Certainty

Learning objectives

At the end of the class you should be able to:

- state the assumptions of deterministic planning
- represent a problem using both STRIPs and the
- feature-based representation of actions.
- specify the search tree for a forward planner
- specify the search tree for a regression planner
- represent a planning problem as a CSP
- Partial-Order Planner

Planning

- Planning is deciding what to do based on an agent’s ability, its goals, and the state of the world.
- 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.
    - Aim find a sequence of actions to solve a goal.

Representing States, Actions, and Goals

- 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.

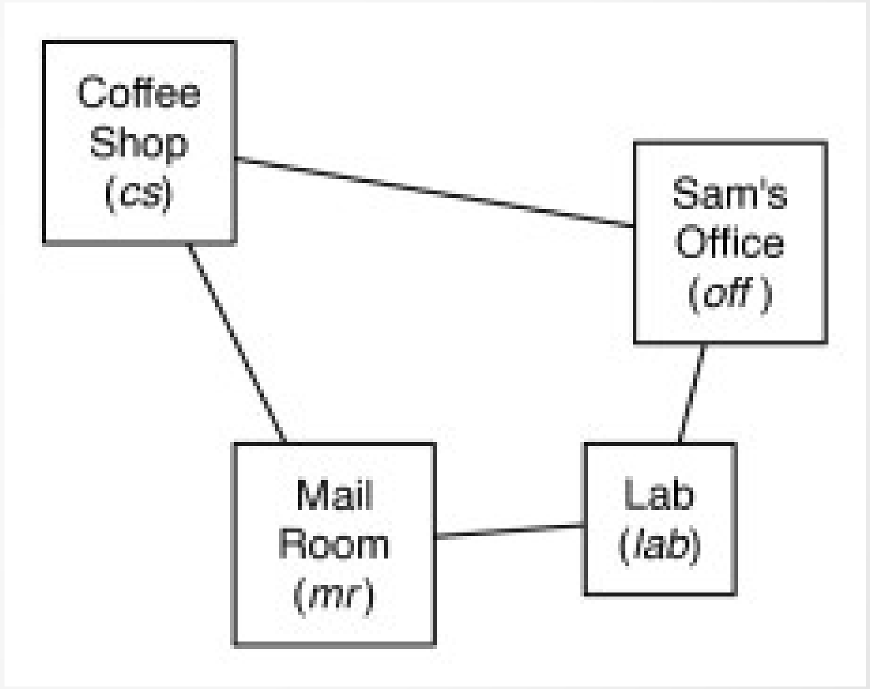
Delivery Robot Example

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



Representing Actions

- State-space Representation
- Feature-based Representation
- STRIPS Representation

Explicit State-space Representation

State	action	Resulting state
s7	act45	s95
s7	act14	s83
s95	act5	s33
...	...	...

state	action	Resulting state
<lab, ¬rhc, swc, ¬mw , rhm>	mc	<mr , ¬rhc, swc, ¬mw , rhm>
<lab, ¬rhc, swc, ¬mw , rhm>	mcc	<off , ¬rhc, swc, ¬mw , rhm>
<off , ¬rhc, swc, ¬mw , rhm>	dm	<off , ¬rhc, ¬swc, ¬mw , ¬rhm>
<off , ¬rhc, swc, ¬mw , rhm>	mcc	<cs, ¬rhc, swc, ¬mw , rhm>
<off , ¬rhc, swc, ¬mw , rhm>	mc	<lab, ¬rhc, swc, ¬mw , rhm>
...	...	...

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"

$rhc' \leftarrow rhc \land Act \neq dc$ (frame rule)

$rhc' \leftarrow Act=puc$(causal rule)

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]

Initial States and Goals

- In a typical planning problem, where the world is fully observable and deterministic, the initial state is defined by specifying the value for each feature for the initial state.
- There are several different kinds of goals:
    - An **achievement goal** is a proposition that must be true in the final state.
    - A **maintenance goal** is a proposition that must be true in every state through which the agent passes. These are often safety goals – the goal of staying away from bad states.
    - A **transient goal** is a proposition that must be achieved somewhere in the plan.
    - A **resource goal** is the goal of minimizing some resource in the plan. For example, the goal may be to minimize fuel consumption or the time required to execute the plan.

Planning

- Forward Planning
- Regression Planning
- Planning as a CSP

Planning

Given:

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

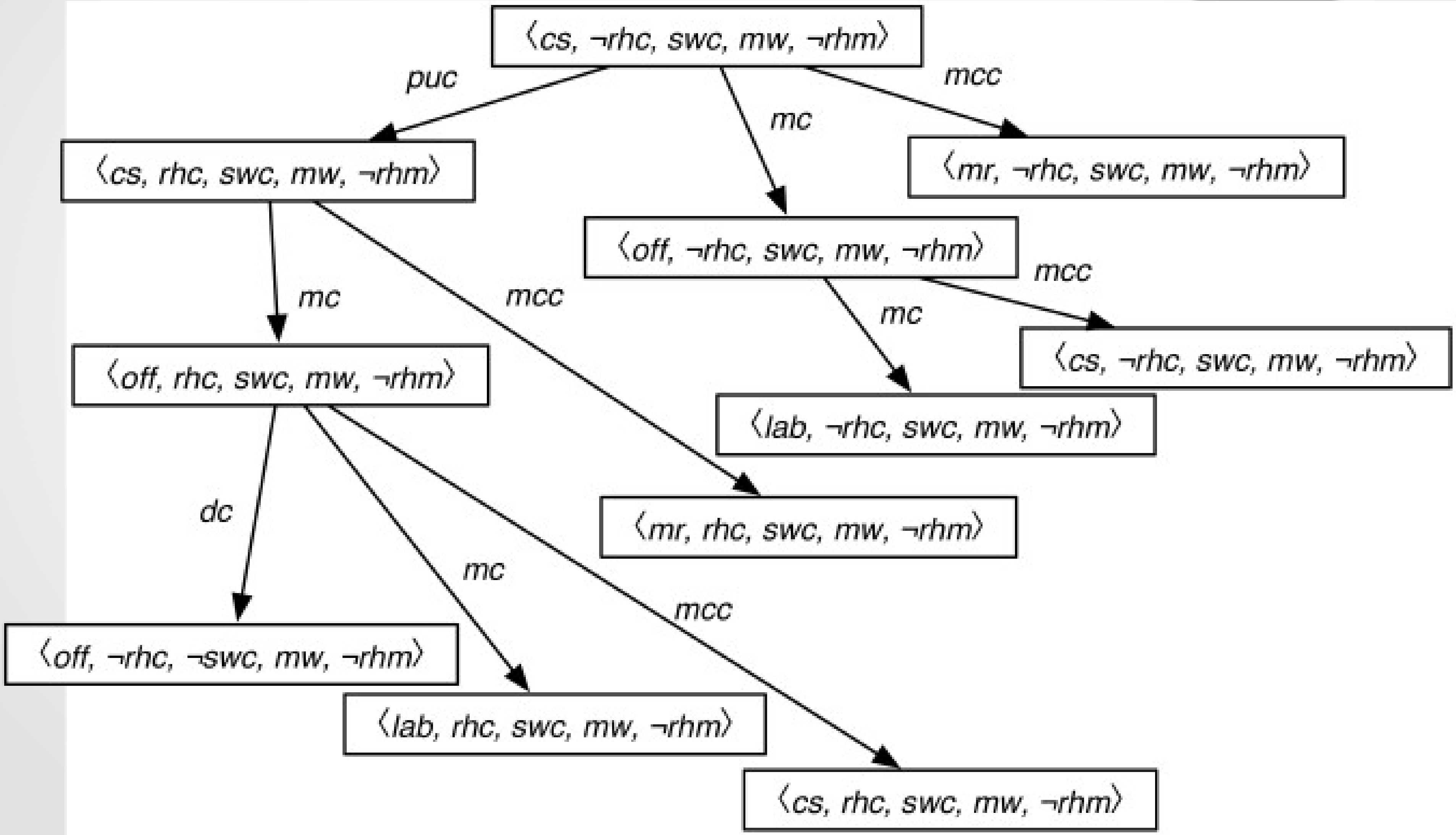
find a sequence of actions that is possible and will result in a state satisfying the goal.

Forward Planning

Idea: search in the state-space graph.

- The nodes represent the states
- The arcs correspond to the actions: The arcs 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.

Example state-space graph


⟨cs, ¬rhc, swc, mw, ¬rhm⟩
puc
mc
mcc
⟨cs, rhc, swc, mw, ¬rhm⟩
⟨mr, ¬rhc, swc, mw, ¬rhm⟩
⟨off, ¬rhc, swc, mw, ¬rhm⟩
mcc
mc
mc
mcc
⟨cs, ¬rhc, swc, mw, ¬rhm⟩
⟨off, rhc, swc, mw, ¬rhm⟩
⟨lab, ¬rhc, swc, mw, ¬rhm⟩
dc
⟨mr, rhc, swc, mw, ¬rhm⟩
mc
mcc
⟨off, ¬rhc, ¬swc, mw, ¬rhm⟩
⟨lab, rhc, swc, mw, ¬rhm⟩
⟨cs, rhc, swc, mw, ¬rhm⟩

Forward planning representation

- The search graph can be constructed on demand: it only constructs reachable states.
- To do a cycle check or multiple path-pruning, the planner need to be able to find repeated states.
- There are a number of ways to represent states:

Forward planning representation

- The search graph can be constructed on demand: it only constructs reachable states.
- To do a cycle check or multiple path-pruning, the planner need to be able to find repeated states.
- There are a number of ways to represent states:
    - As a specification of the value of every feature
    - As a path from the start state

Improving Search Efficiency

Forward search can use domain-specific knowledge specified as:

- a heuristic function that estimates the cost of achieving a goal
- domain-specific pruning of neighbors:
    - don’t go to the coffee shop unless “Sam wants coffee” is part of the goal and Rob doesn’t have coffee
    - don’t pick-up coffee unless Sam wants coffee
    - unless the goal involves time constraints, don’t do the “no move” action.

Regression/Backward Planning

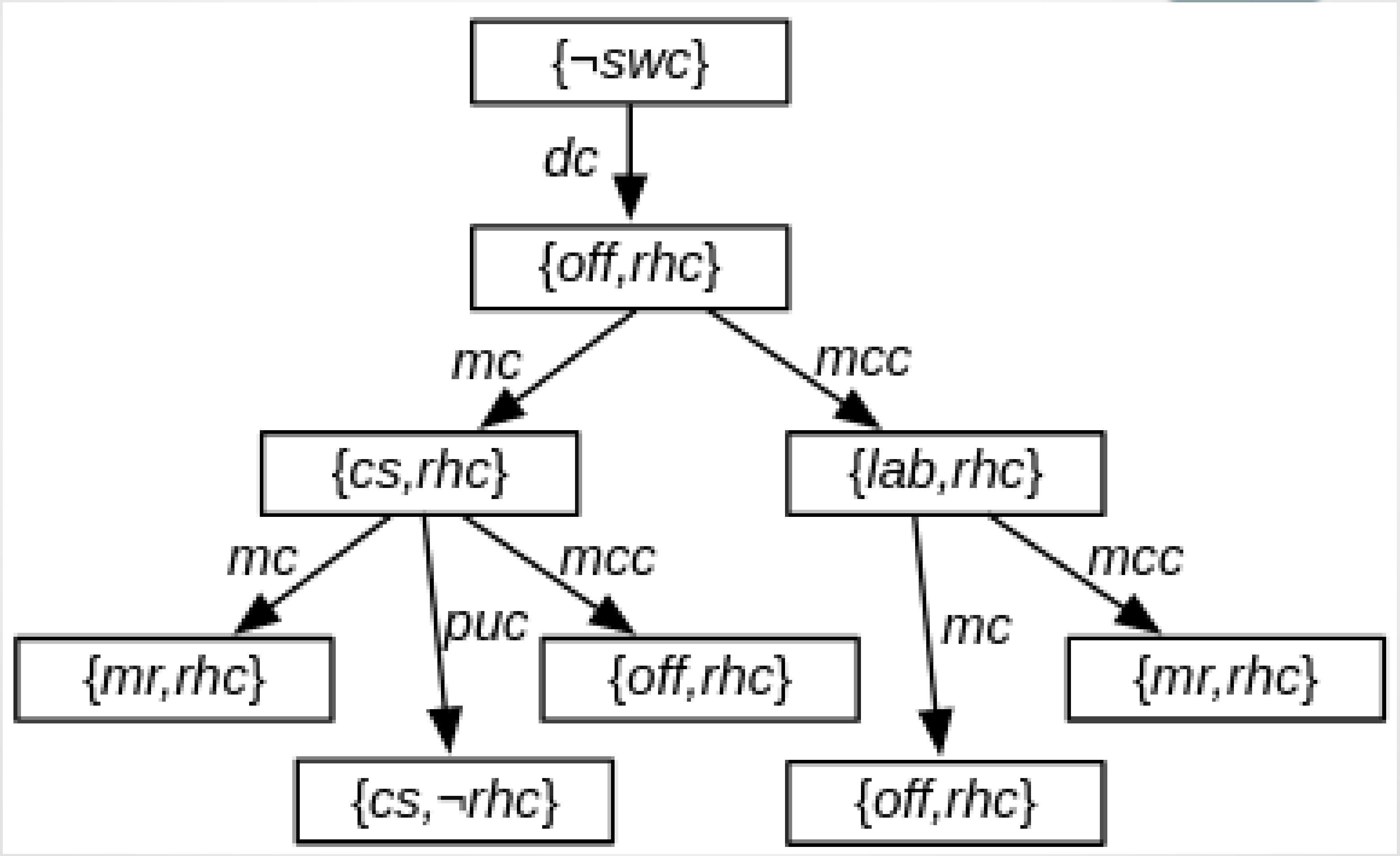
Idea: search backwards from the goal description: nodes correspond to subgoals, and arcs to actions.

- Nodes are propositions: a formula made up of assignments of values to features
- Arcs correspond to actions that can achieve one of the goals
- Neighbors of a node N associated with arc A specify what must be true immediately before A so that N is true immediately after.
- The start node is the goal to be achieved. goal(N) is true if N is a proposition that is true of the initial state.

Defining nodes and arcs

- A node N can be represented as a set of assignments of values to variables:
  - $[X_1 = v_1 , . . . , X_n = v_n]$
  - This is a set of assignments you want to hold.
- The last action is one that achieves one of the $X_i = v_i$,
- and does not achieve $X_j = v'_j$ where v'_j is different to v_j .
- The neighbor of N along arc A must contain:
  - The prerequisites of action A
  - All of the elements of N that were not achieved by A
  - N must be consistent.

Regression example



Formalizing arcs using STRIPS notation

<G , A, N>

where G is $[X_1 = v_1 , . . . , X_n = v_n]$ is an arc if

- $\exists i\ X_i = v_i$ is on the effects list of action A
- $\forall j\ X_j = v'_j$ is not on the effects list for A, where $v'_j \neq v'_j$
- N is preconditions(A) $\cup \{X_k = v_k : X_k = v_k$ *not in* effects(A)$\}$
- and N is consistent in that it does not assign different
- values to any variable.

Loop detection and multiple-path pruning

- Goal G_1 is simpler than goal G_2 if G_1 is a subset of G_2 .
    - It is easier to solve [cs] than [cs, rhc].
- If you have a path to node N have already found a path to a simpler goal, you can prune the path N.

Improving Efficiency

- A search can use a heuristic function that estimates the cost of solving a goal from the initial state.
- You can use domain-specific knowledge to remove impossible goals.
  - E.g., it is often not obvious from an action description to conclude that an agent can only hold one item at any time.

Comparing forward and regression planners

- Which is more efficient depends on:
  - The branching factor
  - 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 a CSP

- Search over planning horizons.
- For each planning horizon, create a CSP constraining possible actions and features
- Also factor actions into action features.

Action Features

- PUC : Boolean variable, the agent picks up coffee.
- DelC : Boolean variable, the agent delivers coffee.
- PUM: Boolean variable, the agent picks up mail.
- DelM: Boolean variable, the agent delivers mail.
- Move: variable with domain {mc, mac, nm} specifies whether the agent moves clockwise, anti-clockwise or doesn’t move

CSP Variables

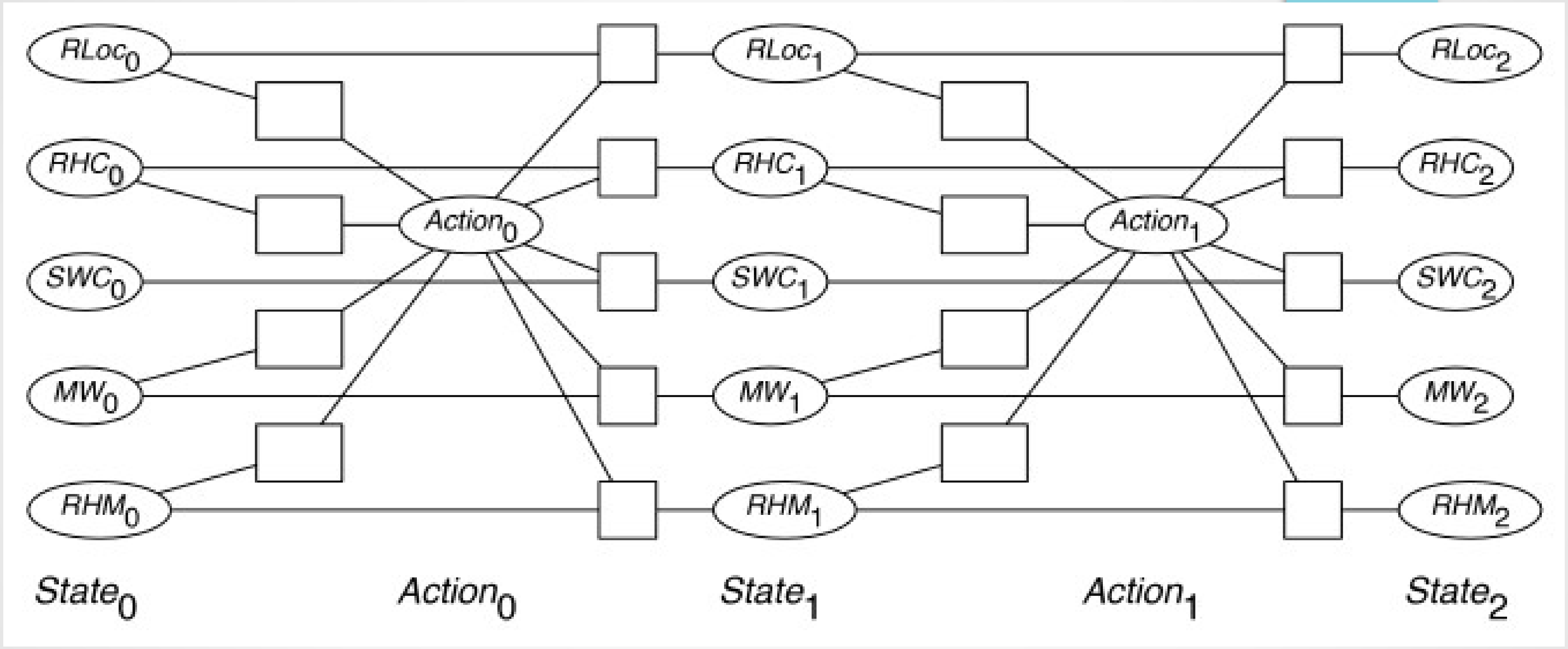
Choose a planning horizon k.

- Create a variable for each state feature and each time from 0 to k.
- Create a variable for each action feature for each time in the range 0 to k − 1.

Constraints

- state constraints that are constraints between variables at the same time step.
- precondition constraints between state variables at time t and action variables at time t that specify constraints on what actions are available from a state.
- effect constraints between state variables at time t, action variables at time t and state variables at time t + 1.
- action constraints that specify which actions cannot co-occur. These are sometimes called mutual exclusion or mutex constraints.
- initial state constraints that are usually domain constraints on the initial state (at time 0).
- goal constraints that constrains the final state to be a state that satisfies the goals that are to be achieved.

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

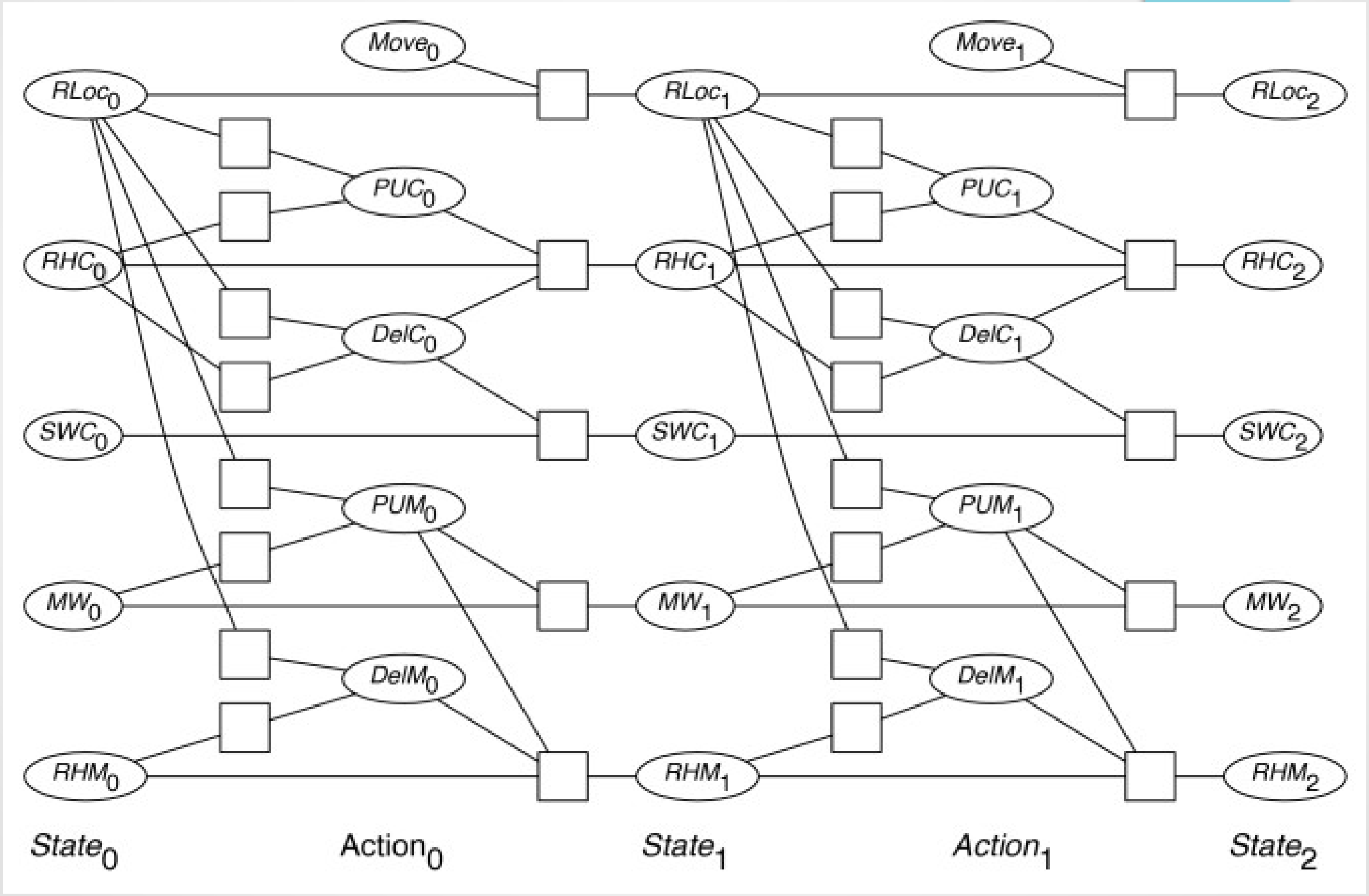
Effect Constraint

RHCi	DCi	PUCi	RHCi+1
true	true	true	true
true	true	false	false
true	false	true	true
true	false	false	true
false	true	true	true
false	true	false	false
false	false	true	true
false	false	false	false

Action Features

- The features representing actions are called action features and
- the features representing states are called state features.
- The action features can be considered as actions in themselves that are carried out in the same time step.
- In this case, there can be an extra set of constraints called action constraints to specify which action features cannot co-occur.
- These are sometimes called mutual exclusion or mutex constraints.

Example


Move0
Move1
RLoc0
RLoc1
RLoc2
PUC0
PUC1
RHC0
RHC1
RHC2
DelC0
DelC1
SWC0
SWC1
SWC2
PUM0
PUM1
MW0
MW1
MW2
DelM0
DelM1
RHM0
RHM1
RHM2
State0
Action0
State1
Action1
State2

Example

Consider finding a plan to get Sam coffee, where initially, Sam wants coffee but the robot does not have coffee. The initial state can be represented as two domain constraints: SWC0=true and on RHC0=false. The goal is that Sam no longer wants coffee, SWCk=false.

With a planning horizon of 2, the CSP is shown in the figure in the previous slide. The goal is represented as the domain constraint SWC2=false, there is a solution RLoc0=cs (the robot has to start in the coffee shop), PUC0=true (the robot has to pick up coffee initially), Move0=mc (the robot has to move to the office), and DC1=true (the robot has to deliver coffee at time 1).

Note that in the representation without factored actions, the problem cannot be solved with a horizon of 2; it requires a horizon of 3, as there are no concurrent actions.

Partial-Order Planning

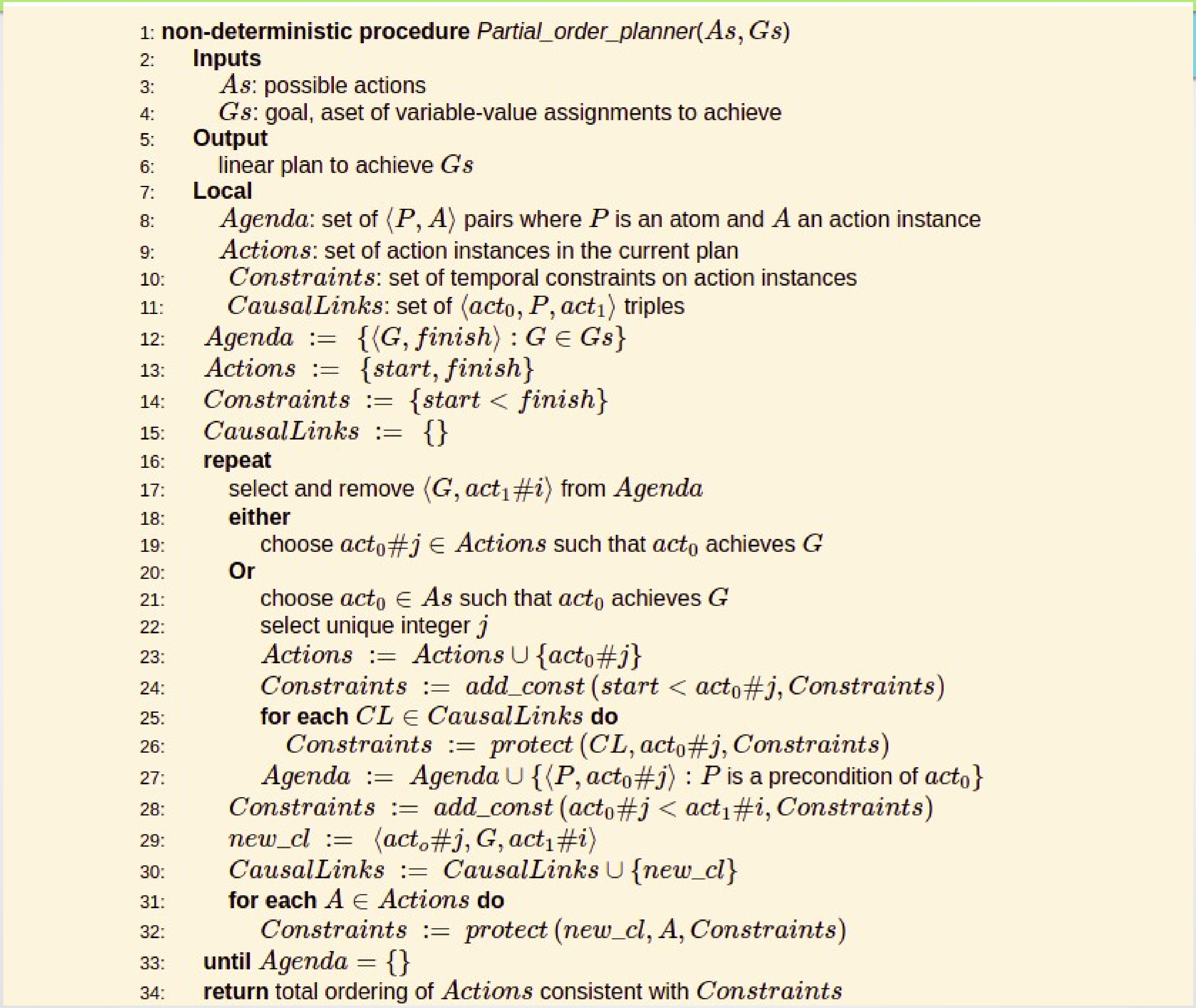
- A partial-order planner maintains a partial ordering between actions and only commits to an ordering between actions when forced. This is sometimes also called a non-linear planner, which is a misnomer because such planners often produce a linear plan.
- Because the same action may be used a number of times in the same plan, for example, the robot may need to move clockwise a number of times, the partial ordering will be between action instances, where an action instance is just a pair of an action and an integer

- A **partial ordering** is a **binary relation** that is transitive and asymmetric.
- A partial-order plan is a set of action instances together with a partial ordering between them, representing a “before” relation on action instances.
- Write act0<act1 if action instance act0 is before action instance act1 in the partial order. This means that the action of act0 must occur before the action of act1.
- The aim of the planner is to produce a partial ordering of the action instances so that any total ordering consistent with the partial ordering will solve the goal from the initial state.
- There are two special action instances, start, that achieves the relations that are true in the initial state, and finish, whose precondition is the goal to be solved.
- Every other action instance is after start and before finish in the partial ordering. The use of these as action instances means that the algorithm does not require special cases for the initial situation and for the goals.
- When the preconditions of finish are achieved, the goal is solved.

- A **causal link** is a triple ⟨act0,P,act1⟩, where act0 and act1 are action instances and P is a Var=val assignment that is in the precondition of act1, and in the effect of act0.
- This means that act0 makes P hold for act1. With this causal link, any other action instance that makes P false must either be before act0 or after act1.

- Informally, a partial-order planner works as follows: Begin with the action instances start and finish and the partial order start<finish.
- The planner maintains an agenda that is a set of ⟨P,A⟩ pairs, where A is an action instance in the plan and P is a variable-value assignment that is a precondition of A that remains to be achieved. Initially, the agenda contains pairs ⟨G,finish⟩, where G is an assignment that must be true in the goal state.
- At each stage in the planning process, a pair ⟨G,act1⟩ is chosen from the agenda, where P is in the precondition for action instance act1.
- Then an action instance, act0, is chosen to achieve P. That action instance is either already in the plan – it could be the start action, for example – or it is a new action instance that is added to the plan.
- Action instance act0 must happen before act1 in the partial order.
- The planner adds a causal link that records that act0 achieves P for action act1. Any action in the plan that makes P false must happen either before act0 or after act1. If act0 is a new action, its preconditions are added to the agenda, and the process continues until the agenda is empty.

Partial-Order Planner



```
non-deterministic procedure Partial_order_planner($As, Gs$)
    Inputs
        $As$: possible actions
        $Gs$: goal, aset of variable-value assignments to achieve
    Output
        linear plan to achieve $Gs$
    Local
        $Agenda$: set of $\langle P, A\rangle$ pairs where $P$ is an atom and $A$ an action instance
        $Actions$: set of action instances in the current plan
        $Constraints$: set of temporal constraints on action instances
        $CausalLinks$: set of $\langle act_0, P, act_1\rangle$ triples
    $Agenda := \{\langle G, finish\rangle : G \in Gs\}$
    $Actions := \{start, finish\}$
    $Constraints := \{start < finish\}$
    $CausalLinks := \{\}$
    repeat
        select and remove $\langle G, act_1\#i\rangle$ from $Agenda$
        either
            choose $act_0\#j \in Actions$ such that $act_0$ achieves $G$
        Or
            choose $act_0 \in As$ such that $act_0$ achieves $G$
            select unique integer $j$
            $Actions := Actions \cup \{act_0\#j\}$
            $Constraints := add\_const(start < act_0\#j, Constraints)$
            for each $CL \in CausalLinks$ do
                $Constraints := protect(CL, act_0\#j, Constraints)$
            $Agenda := Agenda \cup \{\langle P, act_0\#j\rangle : P$ is a precondition of $act_0\}$
        $Constraints := add\_const(act_0\#j < act_1\#i, Constraints)$
        $new\_cl := \langle act_o\#j, G, act_1\#i\rangle$
        $CausalLinks := CausalLinks \cup \{new\_cl\}$
        for each $A \in Actions$ do
            $Constraints := protect(new\_cl, A, Constraints)$
    until $Agenda = \{\}$
    return total ordering of $Actions$ consistent with $Constraints$
```

Example

- Consider the goal of Sam not wanting coffee and no mail waiting (i.e., ¬swc ∧ ¬mw), where in the initial state Rob is in the lab, Sam wants coffee, Rob does not have coffee, there is mail waiting and Rob does not have mail, i.e., RLoc=lab, swc, ¬rhc, mw, ¬rhm.
- We will write instances of action Act as Act#n where n is a unique integer.
- Initially the agenda is: {⟨¬swc,finish⟩,⟨¬mw,finish⟩}.

Example

- Suppose ⟨¬swc,finish⟩ is chosen and removed from the agenda
- One action dc to achieve this, insert in the plan dc#6
- Now agenda contains: {⟨off,dc#6⟩,⟨rhc,dc#6⟩, ⟨¬mw,finish⟩}
- Constraints is {start<finish,start<dc#6,dc#6<finish}.
- There is one causal link, ⟨dc#6,¬swc,finish⟩.
- Suppose ⟨¬mw,finish⟩ is chosen from the agenda.
- One action can achieve this, pum, with precondition {mw,RLoc=mr}.
- new action instance, say pum#7.
- The causal link ⟨pum#7,¬mw,finish⟩ is added to the set of causal links;
- ⟨mw,pum#7⟩ and ⟨mr,pum#7⟩ are added to the agenda.
- Suppose ⟨mw,pum#7⟩ is chosen from the agenda.
- The action start achieves mw
- The causal link ⟨start,mw,pum#7⟩ is added to the set of causal links.
- Nothing is added to the agenda.

- At this stage, there is no ordering imposed between dc#6 and pum#7.
- Suppose ⟨off,dc#6⟩ is removed from the agenda.
- two actions that can achieve off: mc_cs with preconditions cs, and mcc_lab with preconditions lab.
- Suppose it chooses the action instance mc_cs#9.
- The causal link ⟨mc_cs#9,off,dc#6⟩ is added.
- The first violation of a causal link occurs when a move action is used to achieve ⟨mr,pum#7⟩.
- This action violates the causal link ⟨mc_cs#9,off,dc#6⟩, and so must happen after dc#6 or before mc_cs#9.
- Eventually, it finds a plan of action instances, such as:
- start;mc_lab#15;pum#7;mc_mr#40;puc#11;mc_cs#9;dc#6;finish