Planning (Chapter 10)

Slides by Svetlana Lazebnik, 9/2016
with modifications by Mark Hasegawa-Johnson, 1/2018

Planning Problems

- Examples: River Crossing, Cargo Delivery, Design-for-Disassembly, Story-Telling, Translation Alignment, Tower of Hanoi
- Planning as a search problem
  - Formulation
  - Computational Complexity: PSPACE > NP
- Algorithms for planning
  - Forward Chaining
  - Backward Chaining
- Polynomial Heuristics that Reduce the Exponent
  - Number of Steps
  - Planning Graph
  - Mutex (Mutually Exclusive) pairs

Example: River Crossing Problems

https://en.wikipedia.org/wiki/River_crossing_puzzle

- A farmer has a fox, a goat, and a bag of beans to get across the river
- His boat will only carry him + one object
- He can't leave the fox with the goat
- He can't leave the goat with the bag of beans

Solution

https://en.wikipedia.org/wiki/River_crossing_puzzle

fgb -----(farmer, goat)----→ fGb

fGb ←-----(farmer)----------

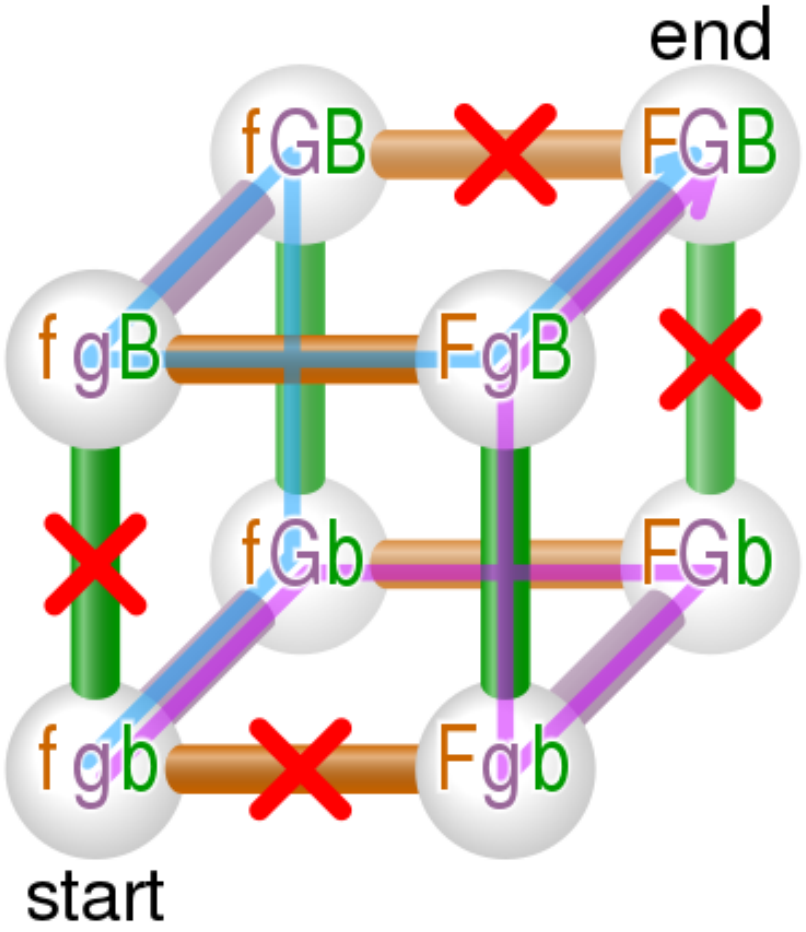
-----(farmer,fox)-----→ FGb

Fgb ←--(farmer,goat)------

-----(farmer,beans)---→ FgB

FgB ←-------(farmer)--------

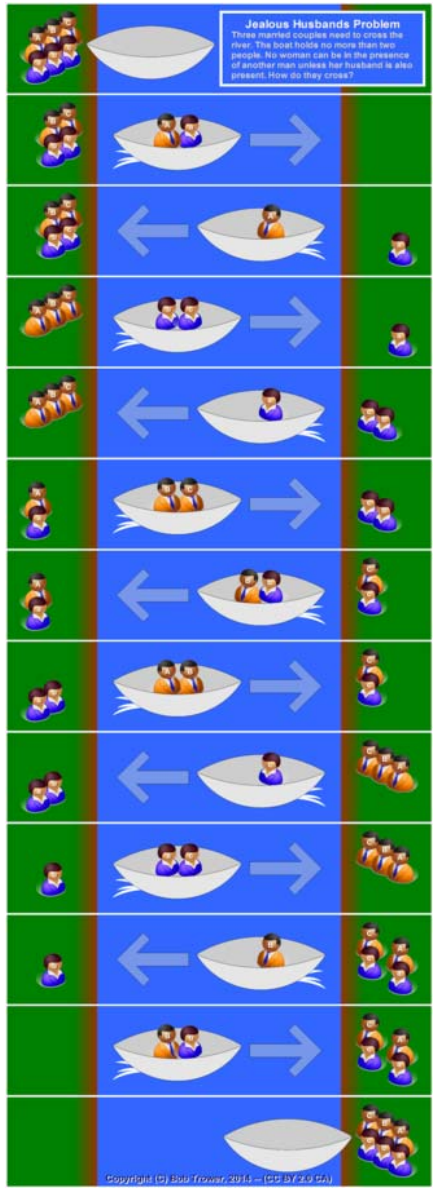
-----(farmer,goat)----→ FGB



Example: Jealous husbands problem

https://en.wikipedia.org/wiki/Missionaries_and_cannibals_problem

- Three couples come to a river. The boat will only hold two people.
- No woman can be alone in the boat, or on either shore, with a man other than her husband.



Example: Cargo delivery problem

- You have packages waiting for pickup at Atlanta, Boston, Charlotte, Denver, Edmonton, and Fairbanks
- They must be delivered to Albuquerque, Baltimore, Chicago, Des Moines, El Paso, and Frisco
- You have two trucks. Each truck can hold only two packages at a time.

Example: Design for Disassembly

”Simultaneous Selective Disassembly and End-of-Life Decision Making for Multiple Products That Share Disassembly Operations,” Sara Behdad, Minjung Kwak, Harrison Kim and Deborah Thurston. *J. Mech. Des* 132(4), 2010, doi:10.1115/1.4001207

- Design decisions limit the sequence in which you can disassemble a product at the end of its life
- Problem statement: design the product in order to make disassembly as cheap as possible

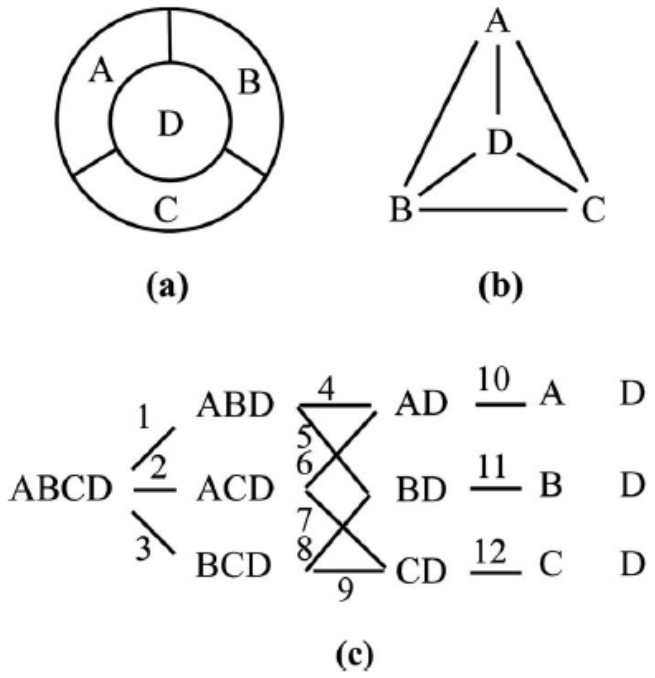
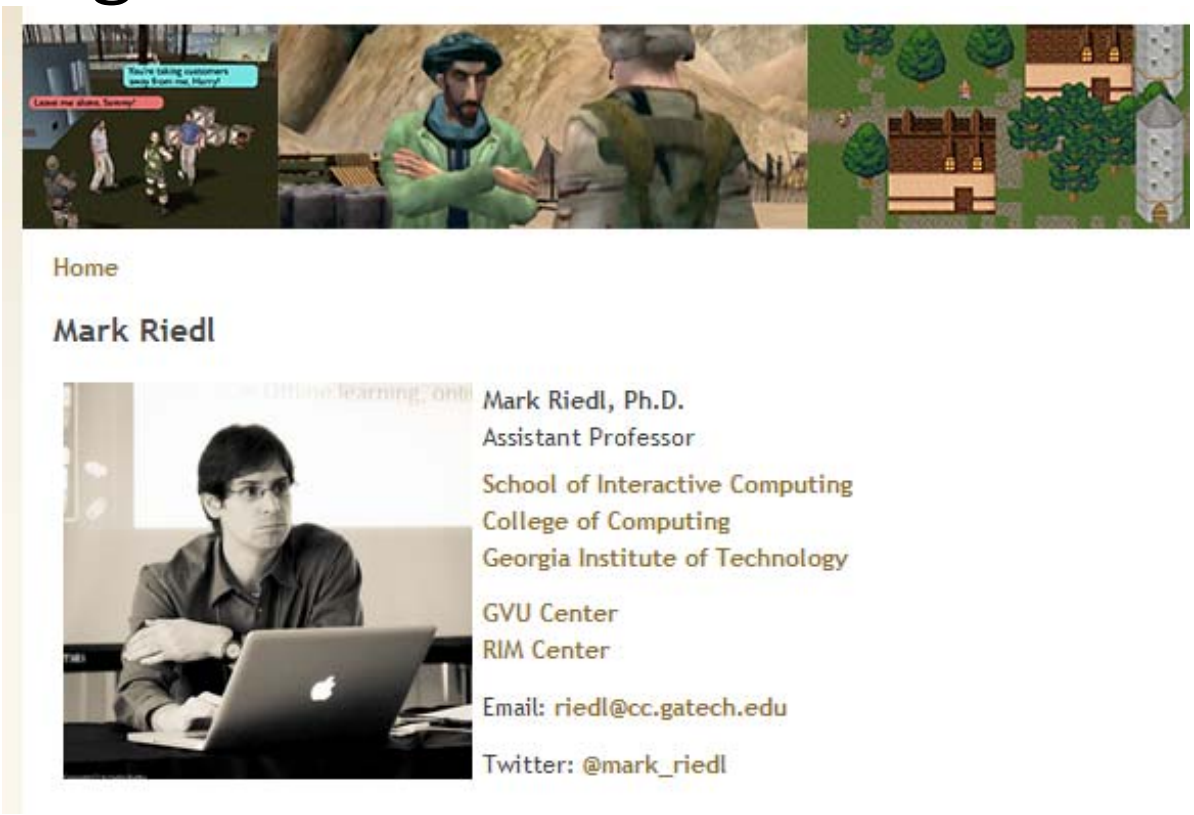


Fig. 1 Simple assembly (*a*), its connection diagram (*b*), and its disassembly graph (*c*) [23]

Application of planning: Automated storytelling



https://research.cc.gatech.edu/inc/mark-riedl

Application of planning: Automated storytelling

- Applications
  - Personalized experience in games
  - Automatically generating training scenarios (e.g., for the army)
  - Therapy for kids with autism
  - Computational study of creativity

https://research.cc.gatech.edu/inc/mark-riedl

Application of planning: the Gale-Church alignment algorithm for machine translation

Table 2
Output from alignment program.

English	French
According to our survey, 1988 sales of mineral water and soft drinks were much higher than in 1987, reflecting the growing popularity of these products. Cola drink manufacturers in particular achieved above-average growth rates.	Quant aux eaux minérales et aux limonades, elles rencontrent toujours plus d'adeptes. En effet, notre sondage fait ressortir des ventes nettement supérieures à celles de 1987, pour les boissons à base de cola notamment.
The higher turnover was largely due to an increase in the sales volume.	La progression des chiffres d'affaires résulte en grande partie de l'accroissement du volume des ventes.
Employment and investment levels also climbed.	L'emploi et les investissements ont également augmenté.

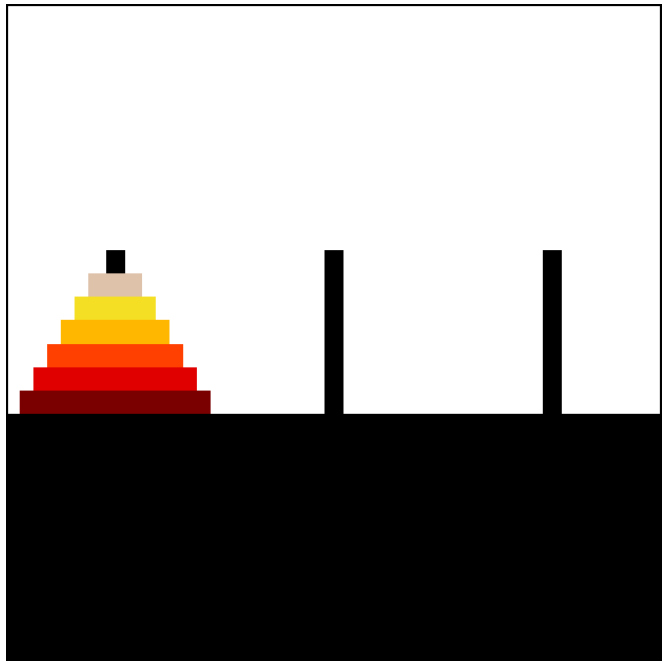
Application of planning: the Gale-Church alignment algorithm for machine translation

1. Let $d(x_1, y_1; 0, 0)$ be the cost of substituting x_1 with y_1,
2. $d(x_1, 0; 0, 0)$ be the cost of deleting x_1,
3. $d(0, y_1; 0, 0)$ be the cost of insertion of y_1,
4. $d(x_1, y_1; x_2, 0)$ be the cost of contracting x_1 and x_2 to y_1,

5. $d(x_1, y_1; 0, y_2)$ be the cost of expanding x_1 to y_1 and y_2, and
6. $d(x_1, y_1; x_2, y_2)$ be the cost of merging x_1 and x_2 and matching with y_1 and y_2.

Example: Tower of Hanoi

https://en.wikipedia.org/wiki/Tower_of_Hanoi



Description	**English:** This is a visualization generated with the walnut based on my implementation at [1] of the iterative algorithm described in Tower of Hanoi
Date	30 April 2015
Source	I designed this using http://thewalnut.io/
Author	Trixx

Planning Problems

- Examples: River Crossing, Cargo Delivery, Design-for-Disassembly, Story-Telling, Translation Alignment, Tower of Hanoi
- Planning as a search problem
  - Formulation
  - Computational Complexity: PSPACE > NP
- Algorithms for planning
  - Forward Chaining
  - Backward Chaining
- Polynomial Heuristics that Reduce the Exponent
  - Number of Steps
  - Planning Graph
  - Mutex (Mutually Exclusive) pairs

Search review

- A search problem is defined by STATES and ACTIONS:
  - States: Initial state, Goal state, Transition model
  - Actions: Transition model, Cost

Planning as Search

States defined by variable=value tuples

- Example: fox=left, goat=right, beans=left
- Variables, Values determined by the problem
  - Usually you have freedom to decide which is which
- Initial state = particular setting of variables to values
- Goal state = desired setting of variables to values
- This is just like CSP!!!

Planning as Search

Pre-specified set of possible actions

- Example: carry_left(beans), carry_right(goat)
- Action = function of one or more variables
- Result = variables changed in pre-defined way
  - With pre-defined cost

- This is not at all like CSP. Order of the actions is important.
  - Constraints apply not just to the goal state, but also to every intermediate state.

A representation for planning

- STRIPS (Stanford Research Institute Problem Solver): classical planning framework from the 1970s
- **States** are specified as conjunctions of predicates
  - Start state: At(home) $\wedge$ Sells(SM, Milk) $\wedge$ Sells(SM, Bananas) $\wedge$ Sells(HW, drill)
  - Goal state: At(home) $\wedge$ Have(Milk) $\wedge$ Have(Banana) $\wedge$ Have(drill)
- **Actions** are described in terms of preconditions and effects:
  - Go(x, y)
    - **Precond:** At(x)
    - **Effect:** ¬At(x) $\wedge$ At(y)
  - Buy(x, store)
    - **Precond:** At(store) $\wedge$ Sells(store, x)
    - **Effect:** Have(x)
- Planning is "just" a search problem

Complexity

- Planning is *PSPACE-complete* > *NP-complete*
  - The computational complexity of finding a plan is exponential
  - The length of the plan is exponential
    - Space necessary to represent it
    - Time necessary to implement it
  - The only thing that's polynomial: the amount of space necessary to represent the world state while finding or implementing a plan
- Example: towers of Hanoi

Complexity of planning

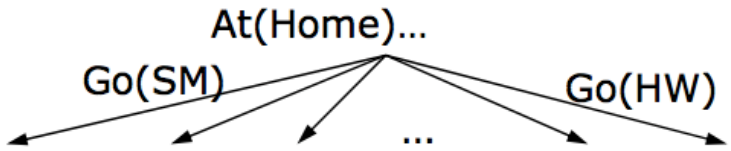
- Planning is *PSPACE-complete*
  - The length of a plan can be exponential in the number of "objects" in the problem!
  - So is game search
- Archetypal PSPACE-complete problem: *quantified boolean formula* (QBF)
  - Example: is this formula true?
    $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \, (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$
- Compare to SAT:
    $\exists x_1 \exists x_2 \exists x_3 \exists x_4 \, (x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_3 \vee \neg x_4)$
- Relationship between SAT and QBF is akin to the relationship between puzzles and games

Planning Problems

- Examples: River Crossing, Cargo Delivery, Design-for-Disassembly, Story-Telling, Translation Alignment, Tower of Hanoi
- Planning as a search problem
  - Formulation
  - Computational Complexity: PSPACE > NP
- Algorithms for planning
  - Forward Chaining
  - Backward Chaining
- Polynomial Heuristics that Reduce the Exponent
  - Number of Steps
  - Planning Graph
  - Mutex (Mutually Exclusive) pairs

Algorithms for planning

- **Forward (progression) state-space search:** starting with the start state, find all applicable actions (actions for which preconditions are satisfied), compute the successor state based on the effects, keep searching until goals are met
  - Can work well with good heuristics



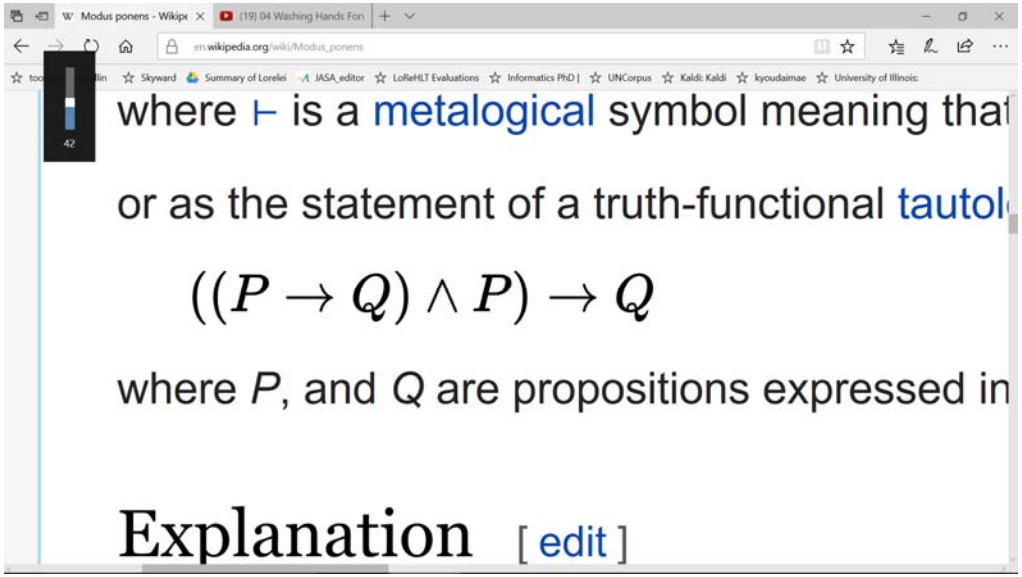
https://www.youtube.com/watch?v=QLNSkFnBYuM

Algorithms for planning

- **Forward (progression) state-space search:** starting with the start state, find all applicable actions (actions for which preconditions are satisfied), compute the successor state based on the effects, keep searching until goals are met
  - Can work well with good heuristics
- **Backward (regression) relevant-states search:** to achieve a goal, what must have been true in the previous state?

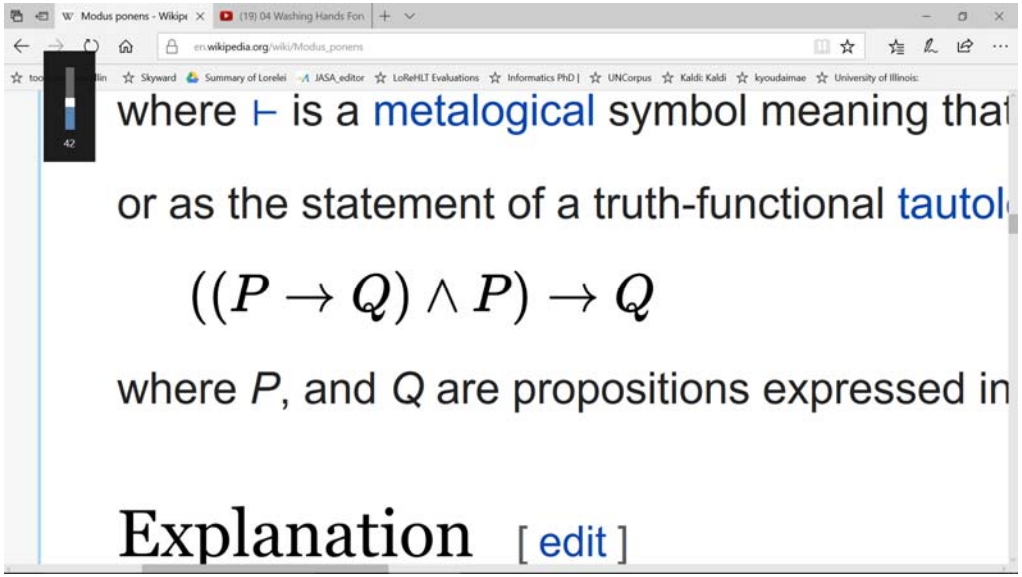
Forward-chaining: each step in the search finds a *possible next state*

- In forward-chaining, we start from a state, P, that the search process has already achieved
- We then check to see whether it is possible to apply *modus ponens*: given the current state P, is there any operation P→Q available?



Backward-chaining: each step in the search finds a *relevant previous state*

- In backward-chaining, we start from a state, Q, that we want to achieve
- We then check to see whether it is possible to apply *modus ponens* in order to generate Q from any other state, P:



Planning Problems

- Examples: River Crossing, Cargo Delivery, Design-for-Disassembly, Story-Telling, Translation Alignment, Tower of Hanoi
- Planning as a search problem
  - Formulation
  - Computational Complexity: PSPACE > NP
- Algorithms for planning
  - Forward Chaining
  - Backward Chaining
- Polynomial Heuristics that Reduce the Exponent
  - Number of Steps
  - Planning Graph
  - Mutex (Mutually Exclusive) pairs

A* Heuristics by Constraint Relaxation

- Heuristics from Constraint Relaxation: The heuristic h(n) is the number of steps it would take to get from n to G, if problem constraints were relaxed --- this guarantees that h(n) is admissible
- $h_1(n)$ dominates $h_2(n)$ ($h_1(n) \geq h_2(n)$) if $h_1(n)$ is computed by relaxing fewer constraints.

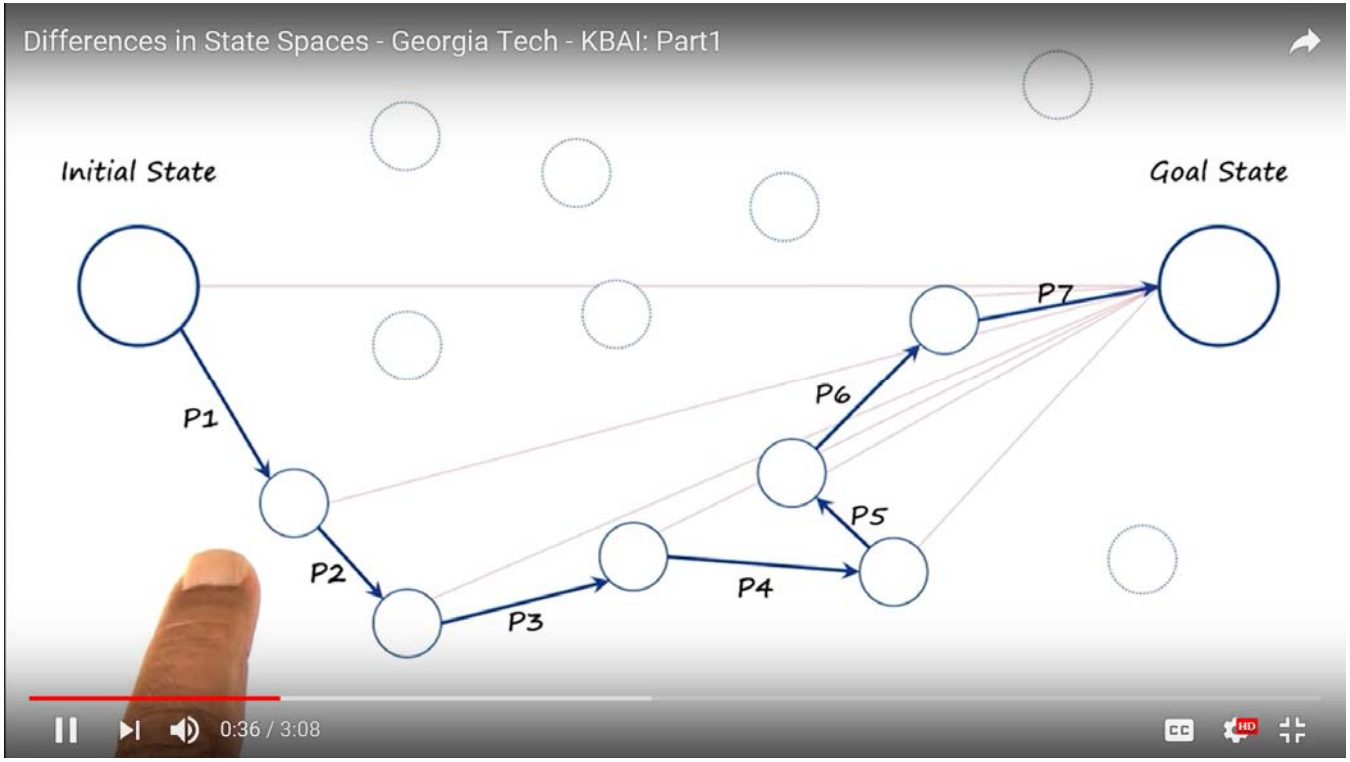
First heuristic: number of goal fluents left to achieve

- Terminology: the *fluent* Xk is defined to be true iff the variable X is equal to the value k, and false otherwise.

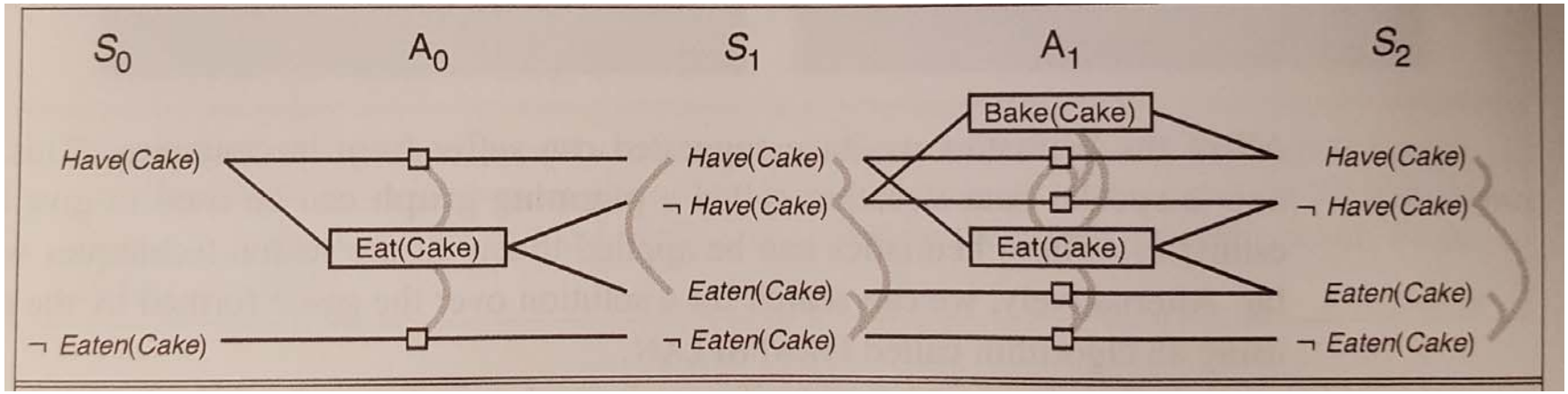
- Heuristic #1: Count the number of actions necessary to generate all of the fluents in the goal state that aren't already true.
  - What got relaxed: we ignore action pre-requisites.

- Example: 6 people on left side of the river, we want 6 people on the right side, we have a 2-person boat. Minimum # actions: h(n) = 3.

Heuristics for planning: means-ends analysis

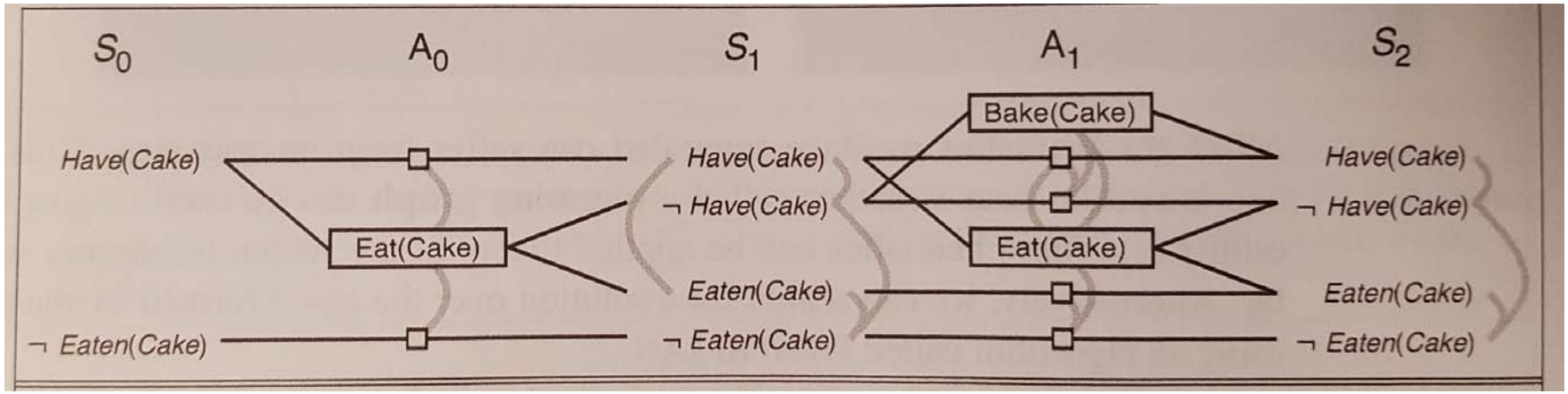


Example problem that we'll use for next two heuristics



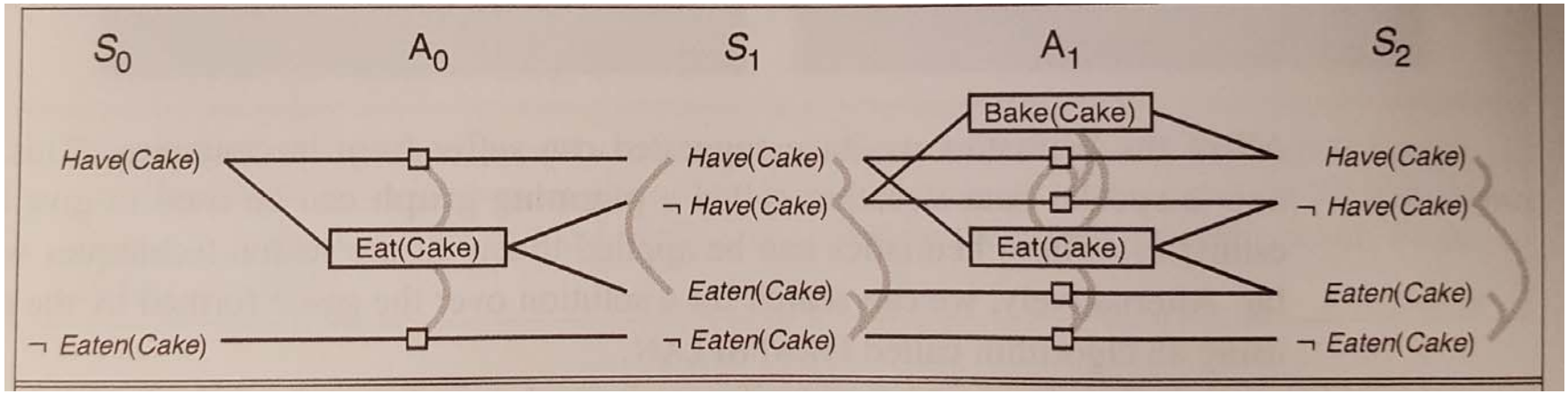
- Example: two actions possible, Eat(Cake) and Bake(Cake)
- Initial state: Have(Cake), ¬Eaten(Cake)
- Goal state: Have(Cake), Eaten(Cake)

Heuristic #2: Get the fluents into the planning graph



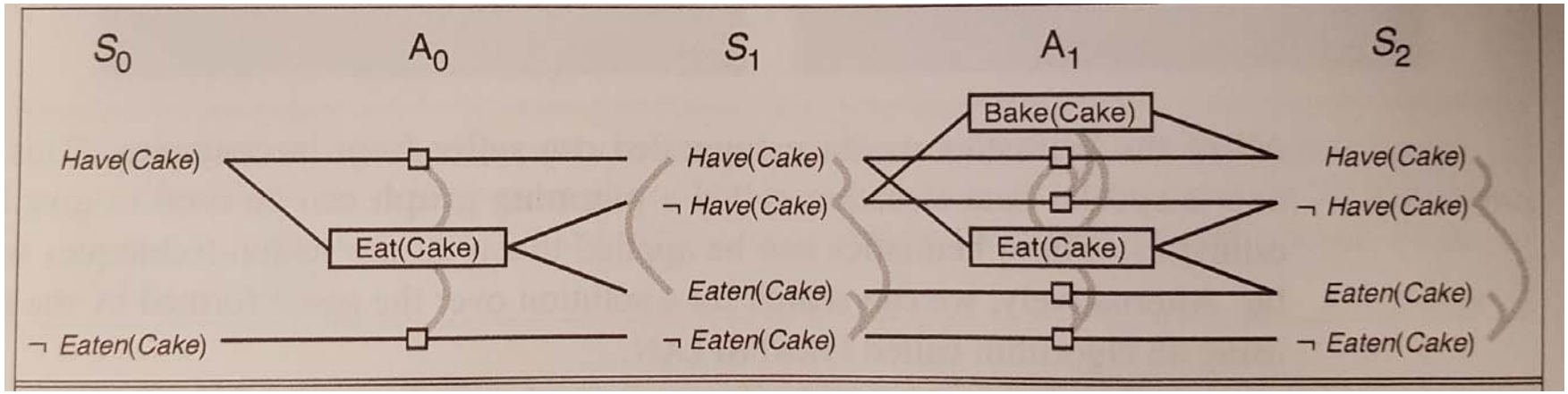
- Planning Graph:
  - At stage S_n, list all of the fluents that COULD BE MADE TO BE TRUE after n actions
  - At stage A_n, list all of the actions whose pre-requisites are available in stage S_n
    - Including the "persist" action, represented above as a small blank square
  - If an action at stage S_n generates fluent F as an output, then fluent F is listed at stage S_{n+1}

Heuristic #2: Get the fluents into the planning graph



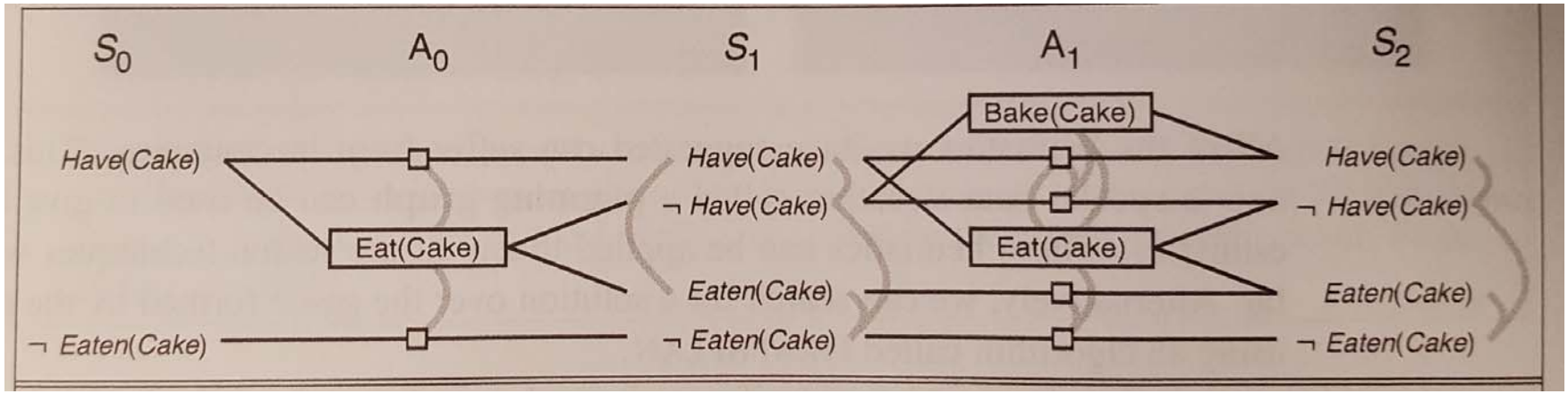
- Convergence of the Planning Graph:
  - Fluents grow: If a fluent is possible at stage S_n, then it is also possible at S_{n+1}
  - Actions grow: If an action is possible at stage A_n, then it is also possible at A_{n+1}

Heuristic #2: Get the fluents into the planning graph



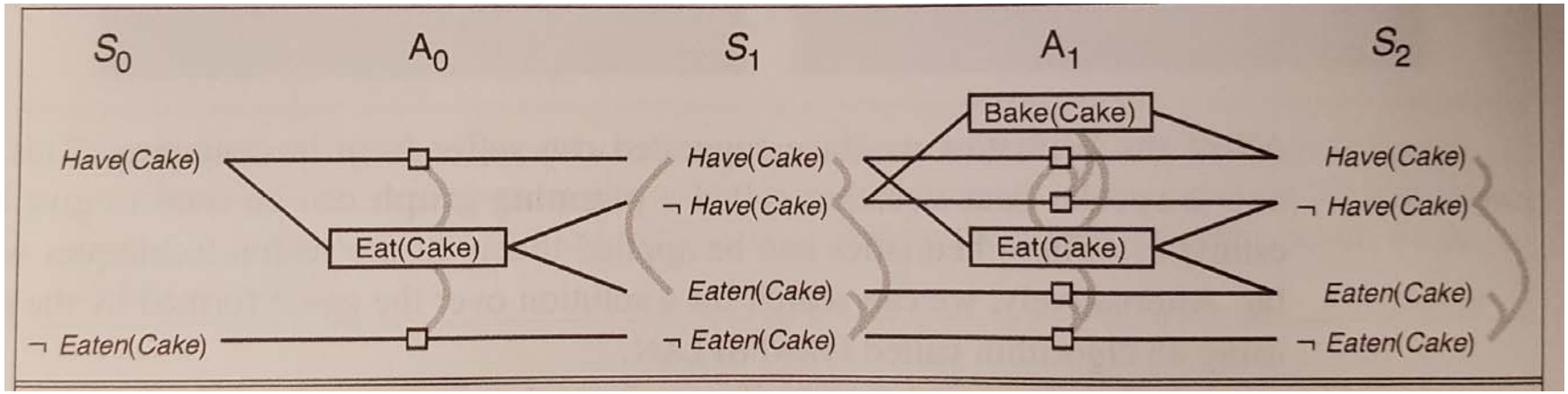
- Heuristic:
  - h(n) = # stages between the current stage and the first stage at which all of the goal-state fluents COULD BE MADE TO BE TRUE
  - "Current stage" can be defined in a few different ways:
    - Based on the number of actions you've already done, … or …
    - The first stage at which all of the fluents in your current world state are already present

Heuristic #3: Mutex links



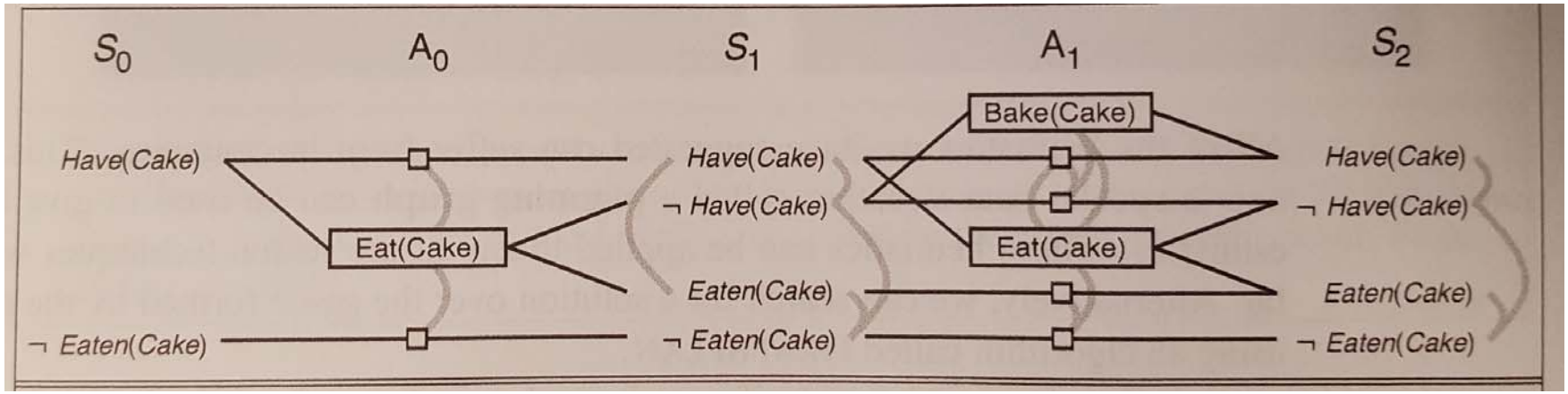
- Mutex (mutually exclusive) link exists between fluents that are polar opposites, F and ¬F
- Mutex exists between actions whose pre-requisite states are mutex, e.g., one requires F, while the other requires ¬F
- Mutex exists between any two fluents, A and B, if every action that generates fluent A is mutex with every action that generates fluent B

Heuristic #3: Get the fluents into the planning graph



- Convergence of the Mutex Links: Mutex links disappear:
  - If two fluents are mutex at stage S_{n+1}, then they are also mutex at S_n
  - If two actions are mutex at stage A_{n+1}, then they are also mutex at A_n

Heuristic #3: Get the fluents into the planning graph



- Heuristic: h(n) = # stages from the current world state until the first stage at which the goal state is non-mutex
- In fact, this is exactly the true path cost, if all constraints are binary

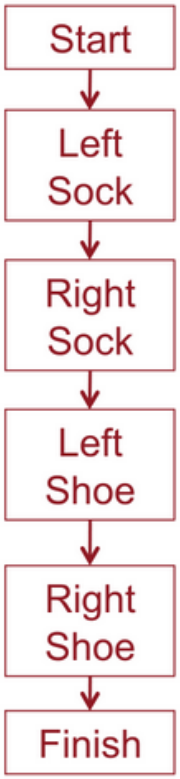
Situation space planning vs. plan space planning

- **Situation space planners:** each node in the search space represents a world state, arcs are actions in the world
  - Plans are sequences of actions from start to finish
  - Must be *totally ordered*
- **Plan space planners:** nodes are (incomplete) plans, arcs are transformations between plans
  - Actions in the plan may be *partially ordered*
  - **Principle of least commitment:** avoid ordering plan steps unless absolutely necessary

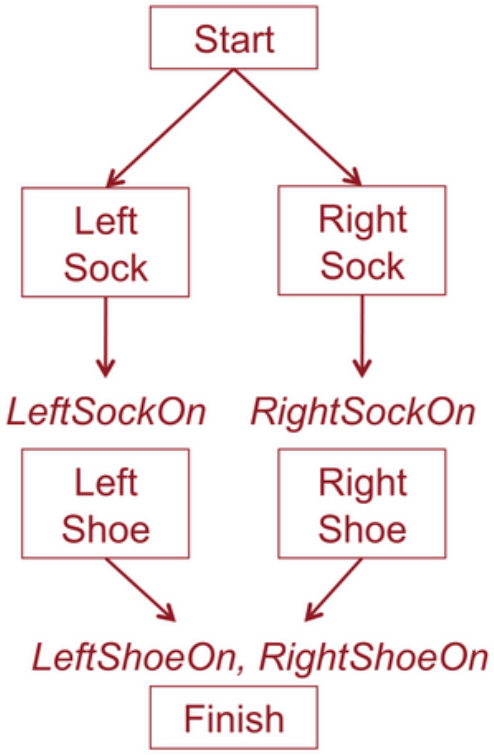
Partial order planning

- Task: put on socks and shoes

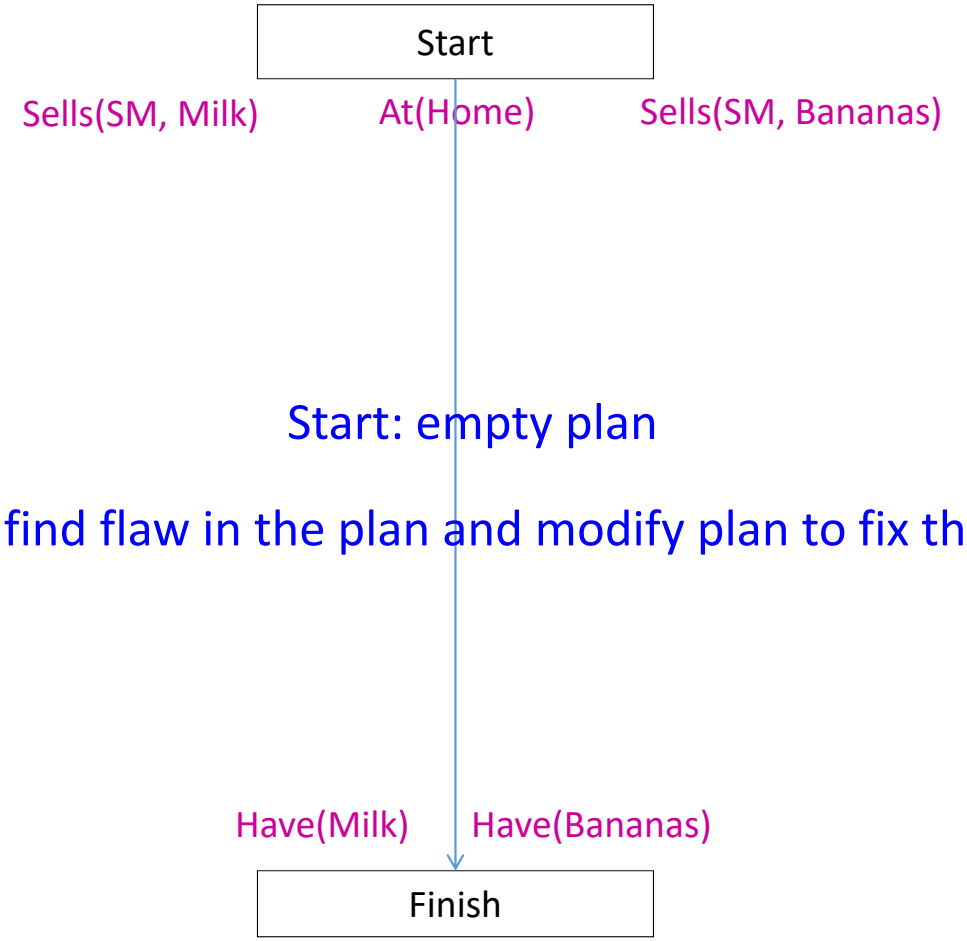
Total order (linear) plans



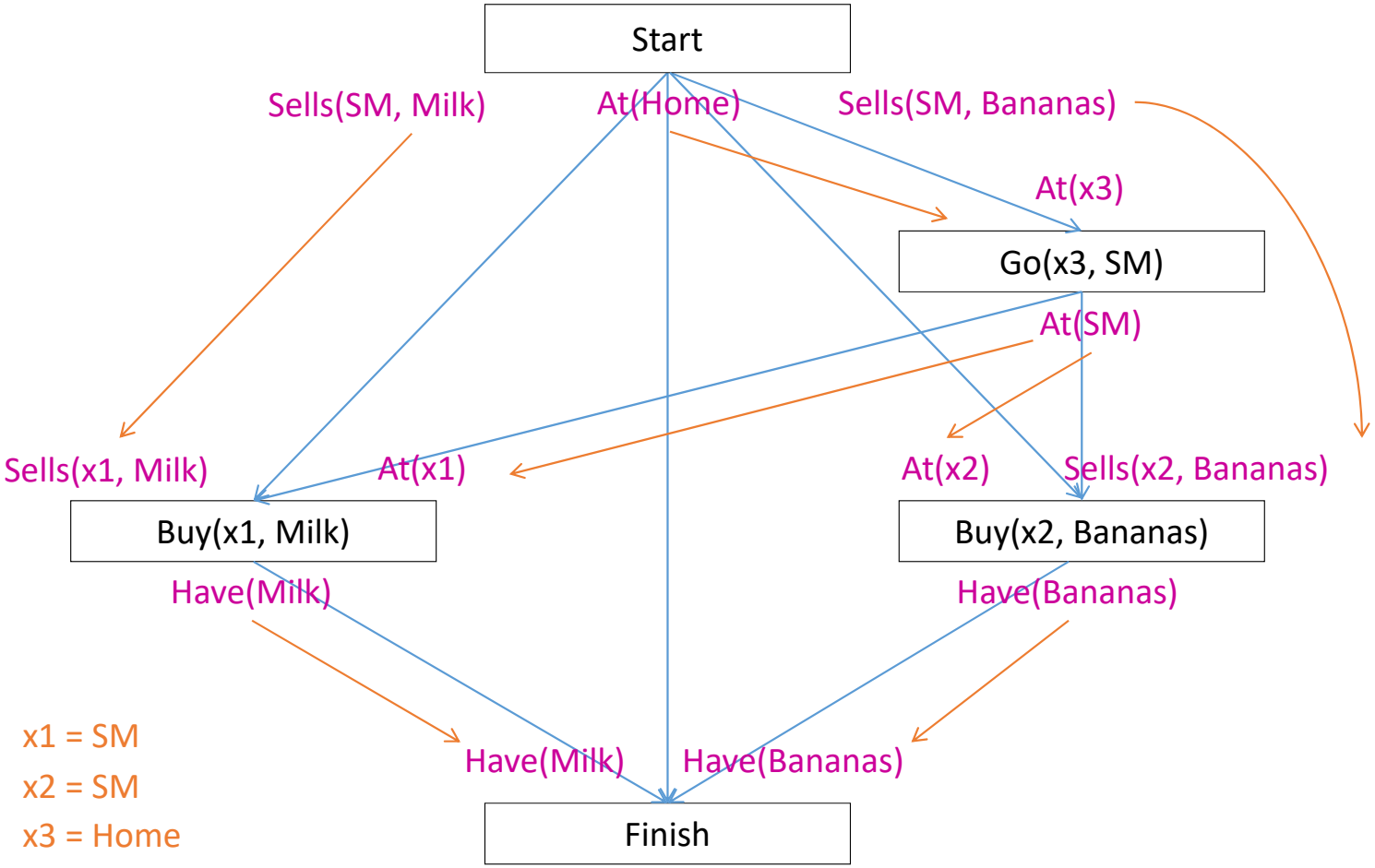
Partial order plan



Partial Order Planning Example



Partial Order Planning Example



Real-world planning

- Resource constraints
  - Instead of “static,” the world is “semidynamic:” we can’t think forever
- Actions at different levels of granularity: hierarchical planning
  - In order to make the depth of the search smaller, we might convert the world from “fully observable” to “partially observable”
- Contingencies: actions failing
  - Instead of being “deterministic,” maybe the world is “stochastic”
- Incorporating sensing and feedback
  - Possibly necessary to address stochastic or multi-agent environments