Set 9: Planning
Classical Planning Systems

ICS 271 Fall 2013

Outline: Planning

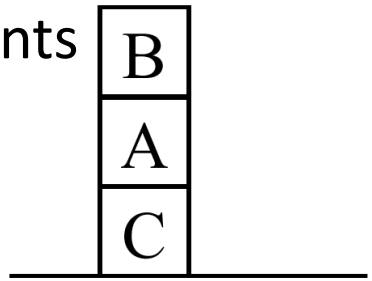
- Classical Planning:
  - Situation calculus
  - PDDL: Planning domain definition language
- STRIPS Planning
- Planning graphs
- Readings: Russel and Norvig chapter 10

The Situation Calculus

- A **goal** can be described by a sentence:
  if we want to have a block on *B* $(\exists x) On(x, B)$
- **Planning**: finding a set of actions to achieve a goal sentence.
- Situation Calculus (McCarthy, Hayes, 1969, Green 1969)
  - A Predicate Calculus formalization of *states*, *actions*, and their *effects*.
  - S_o state in the figure can be described by:

$$On(B, A) \wedge On(A, C) \wedge On(C, Fl) \wedge Clear(B) \wedge clear(Fl)$$

we reify the state and include them as arguments




```
On(B,A)
On(A,C)
On(C,Fl)
Clear(B)
Clear(Fl)
```

The Situation Calculus (continued)

- The atoms denotes relations over states called **fluents**.

$$On(B, A, S_0) \wedge On(A, C, S_0) \wedge On(C, Fl, S_0) \wedge clear(B, S_0)$$

- We can also have.

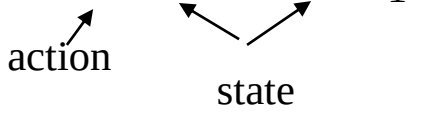
$$(\forall x, y, s)[On(x, y, s) \wedge \neg(y = Fl) \rightarrow \neg Clear(y, s)]$$

$$(\forall s) Clear(Fl, s)$$

- Knowledge about state and actions = predicate calculus theory.
- Inference can be used to answer:
  - Is there a state satisfying a goal?
  - How can the present state be transformed into that state by actions? The answer is a *plan*

Representing Actions

- Reify the actions: denote an action by a symbol
- actions are **functions**
- move(B,A,Fl): move block A from block B to Fl
- move (x,y,z) - action schema
- **do**: A function constant, **do** denotes a function that maps actions and states into states
  - $do(\alpha, \sigma) \rightarrow \sigma_1$



Representing Actions (continued)

- Express the effects of actions.
  - Example: (on, move) (expresses the effect of move on "On")
  - Positive effect axiom:

$$[On(x, y, s) \wedge Clear(x, s) \wedge Clear(z, s) \wedge (x \neq z) \rightarrow On(x, z, do(move(x, y, z), s))]$$

  - Negative effect axiom:

$$[On(x, y, s) \wedge Clear(x, s) \wedge Clear(z, s) \wedge (x \neq z) \rightarrow \neg On(x, y, do(move(x, y, z), s))]$$

  - Positive: describes how action makes a fluent true
  - Negative : describes how action makes a fluent false
  - Antecedent: pre-condition for actions
  - Consequent: how the fluent is changed

Frame Axioms

- Not everything true can be inferred
  On(C,Fl) remains true but cannot be inferred
- Actions have local effect
  - We need **frame axioms** for each action and each fluent that does not change as a result of the action
  - example: frame axioms for (move, on)
  - If a block is on another block and *move* is not relevant, it will stay the same.
    - Positive:

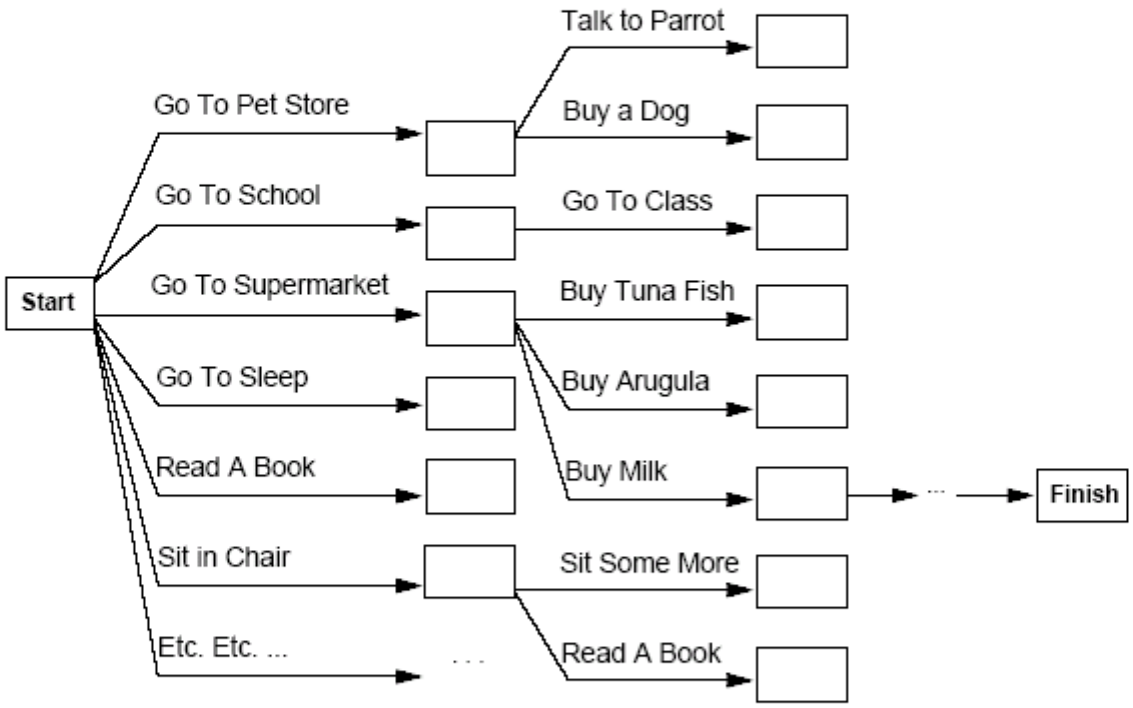
$$[On(x, y, s) \wedge (x \neq u)] \rightarrow On(x, y, do(move(u, v, z), s))$$

    - negative

$$(\neg On(x, y, s) \wedge [(x \neq u) \vee (y \neq z)]) \rightarrow \neg On(x, y, do(move(u, v, z), s)$$

Search vs. planning

Consider the task *get milk, bananas, and a cordless drill*
Standard search algorithms seem to fail miserably:



After-the-fact heuristic/goal test inadequate

Summary so far

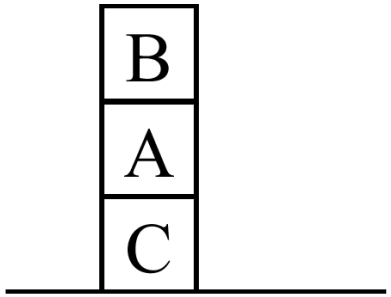
- Situational Calculus
  - Use FOL
    - Initial state
    - Goal state
    - Actions
    - Special symbol – situation
  - Use resolution for inference
- Issues
  - Representation – frame problem
    - Frame axioms
      - #actions X #predicates X #timeslots – positive/negative effects
  - Performance
    - Resolution
- Not widely used

STRIPS Planning systems
PDDL: Planning Domain Definition Language

STRIPS: describing goals and state

Factored representation of states

- On(B,A)
- On(A,C)
- On(C,Fl)
- Clear(B)
- Clear(Fl)




```
On(B,A)
On(A,C)
On(C,Fl)
Clear(B)
Clear(Fl)
```

- The formula describes a set of world states
- Planning search for a formula satisfying a goal description
- **State descriptions**: conjunctions of ground literals.
- Given a goal wff, the search algorithm looks for a sequence of **actions**

That transform into a state description that entails the goal wff.

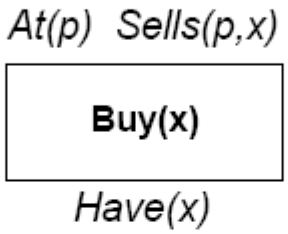
STRIPS Description of Operators

- A STRIPS operator has 3 parts:
  - A set PC, of ground literals (*preconditions*)
  - A set D, of ground literals called the *delete list*
  - A set A, of ground literals called the *add list*
- Usually described by Schema*: Move(x,y,z)*
  - *PC: On(x,y) and Clear(x) and Clear(z)*
  - *D: Clear(z) , On(x,y)*
  - *A: On(x,z), Clear(y), Clear(Fl)*
- A state S_{i+1} is created applying operator O by adding A and deleting D to/from S_i.

STRIPS operators

Tidily arranged actions descriptions, restricted language

ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p, x)$
EFFECT: $Have(x)$

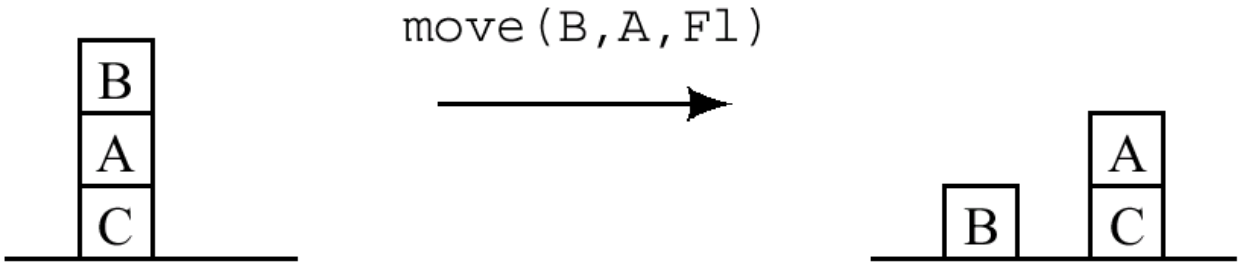


[Note: this abstracts away many important details!]

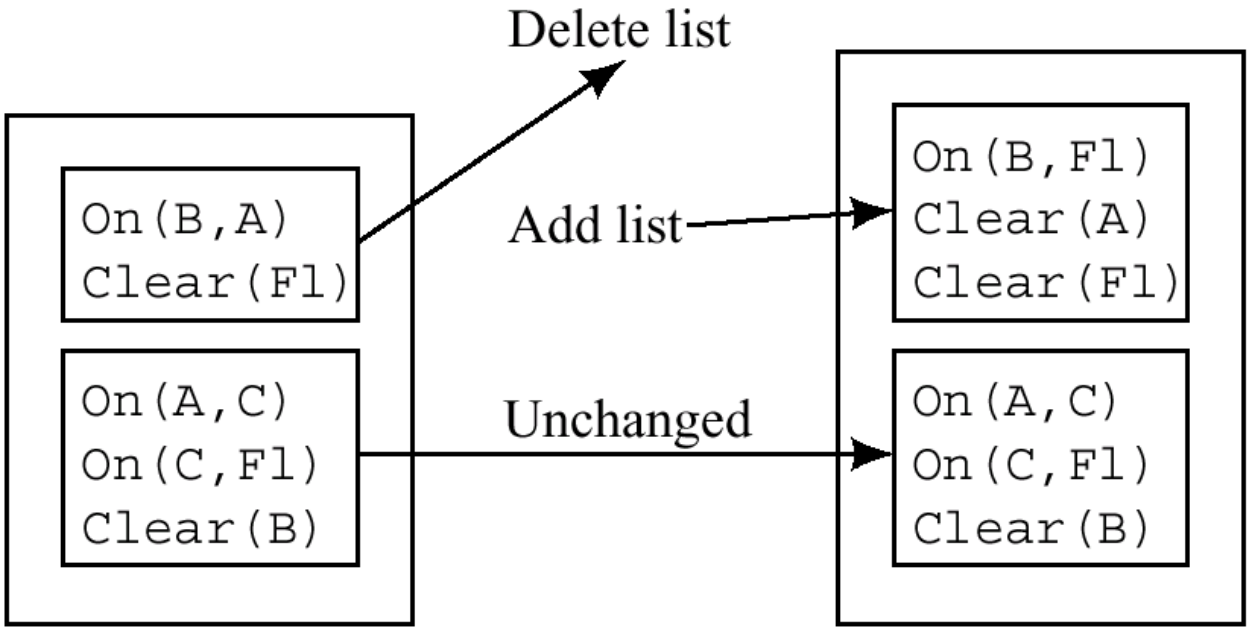
Restricted language $\Rightarrow$ efficient algorithm
- Precondition: conjunction of positive literals
- Effect: conjunction of literals

A complete set of STRIPS operators can be translated into a set of successor-state axioms

Example: the move operator



Precondition:
On(B,A)
Clear(B)
Clear(Fl)



PDDL

- A language that yields a search problem
- A state is a set of ground literals
- Closed world assumption: fluents that are not mentioned are false.
- Action schema:

  Action(Fly(p,from,to),

  Precondition: At(p,from) & Plane(p) & Airport(from) & Airport(to)

  Effect: not At(p,from) & At(p,to)
- *The schema consists of precondition and effect lists*
- *PDDL is very close to STRIP language*

- *A set of action schemas is a definition of a planning domain.*
- *A specific problem is defined by an initial state (a set of ground literals) and a goal: conjunction of literals, some not grounded (At(p,SFO), Plane(p))*

- Both feasible and optimal plan finding is decidable, but in PSPACE

The block world

$Init(On(A, Table) \wedge On(B, Table) \wedge On(C, Table)$
$\quad \wedge Block(A) \wedge Block(B) \wedge Block(C)$
$\quad \wedge Clear(A) \wedge Clear(B) \wedge Clear(C))$
$Goal(On(A, B) \wedge On(B, C))$
$Action(Move(b, x, y),$
$\quad$ PRECOND: $On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge$
$\quad\quad (b \neq x) \wedge (b \neq y) \wedge (x \neq y),$
$\quad$ EFFECT: $On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y))$
$Action(MoveToTable(b, x),$
$\quad$ PRECOND: $On(b, x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x),$
$\quad$ EFFECT: $On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))$

Figure 11.4 A planning problem in the blocks world: building a three-block tower. One solution is the sequence $[Move(B, Table, C), Move(A, Table, B)]$.

A STRIP/PDDL description of an aircargo transportation problem

Problem: flying cargo in planes from one location to another

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND}: At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT}: \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND}: In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT}: At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND}: At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT}: \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Figure 11.2 A STRIPS problem involving transportation of air cargo between airports.

In(c,p)- cargo c is inside plane p
At(x,a) – object x is at airport a

STRIP for spare tire problem

Problem: Changing a flat tire

$Init(At(Flat, Axle) \wedge At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
PRECOND: $At(Spare, Trunk)$
EFFECT: $\neg At(Spare, Trunk) \wedge At(Spare, Ground))$
$Action(Remove(Flat, Axle),$
PRECOND: $At(Flat, Axle)$
EFFECT: $\neg At(Flat, Axle) \wedge At(Flat, Ground))$
$Action(PutOn(Spare, Axle),$
PRECOND: $At(Spare, Ground) \wedge \neg At(Flat, Axle)$
EFFECT: $\neg At(Spare, Ground) \wedge At(Spare, Axle))$
$Action(LeaveOvernight,$
PRECOND:
EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk)$
$\wedge \neg At(Flat, Ground) \wedge \neg At(Flat, Axle))$

Figure 11.3 The simple spare tire problem.

Algorithms for Planning as State-space Search

- Forward (progression) state-space search
- Backward (regression) state-space search
- Heuristic search
  - Relaxed models
  - Goal decomposition
  - Planning graphs

Planning forward and backwards

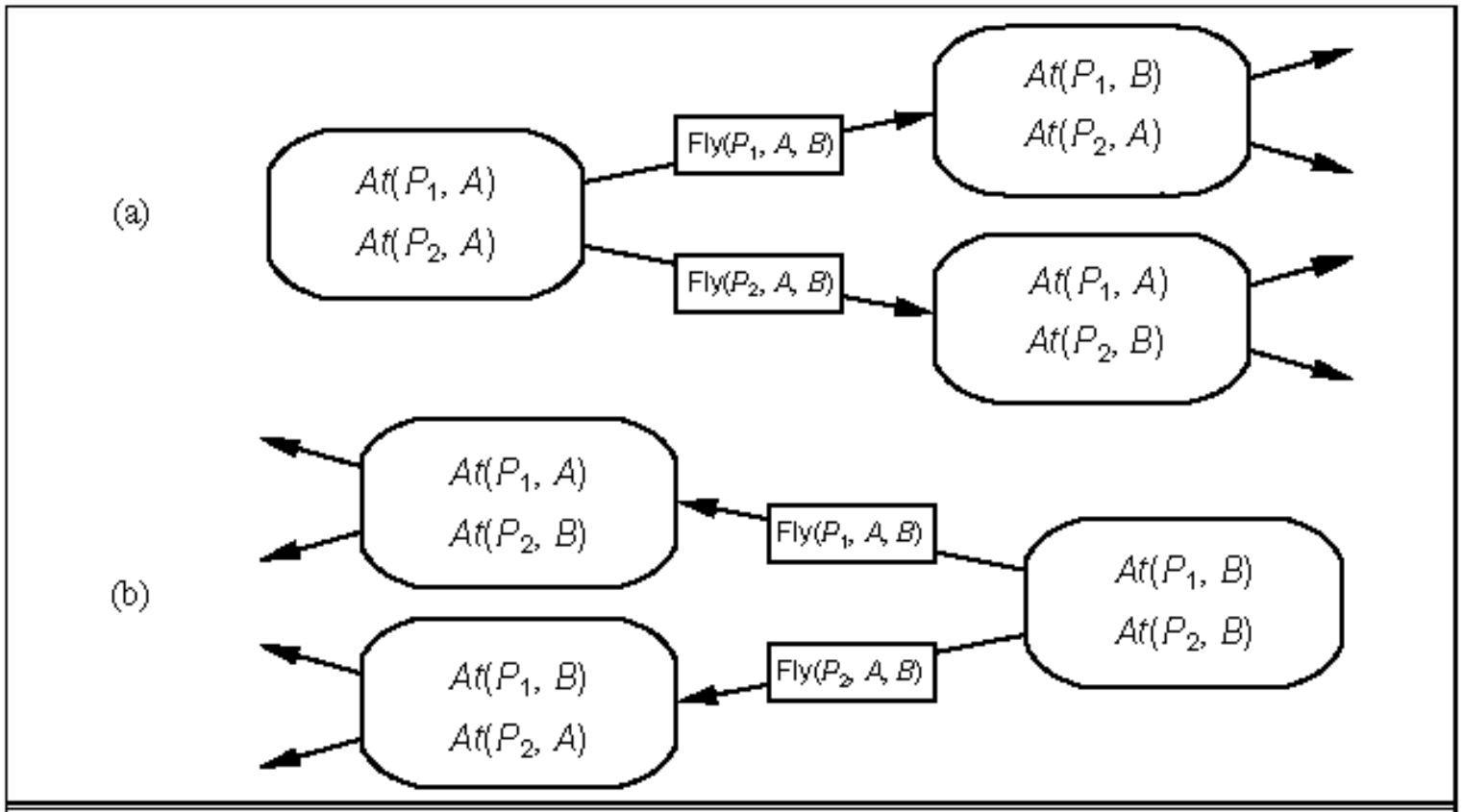
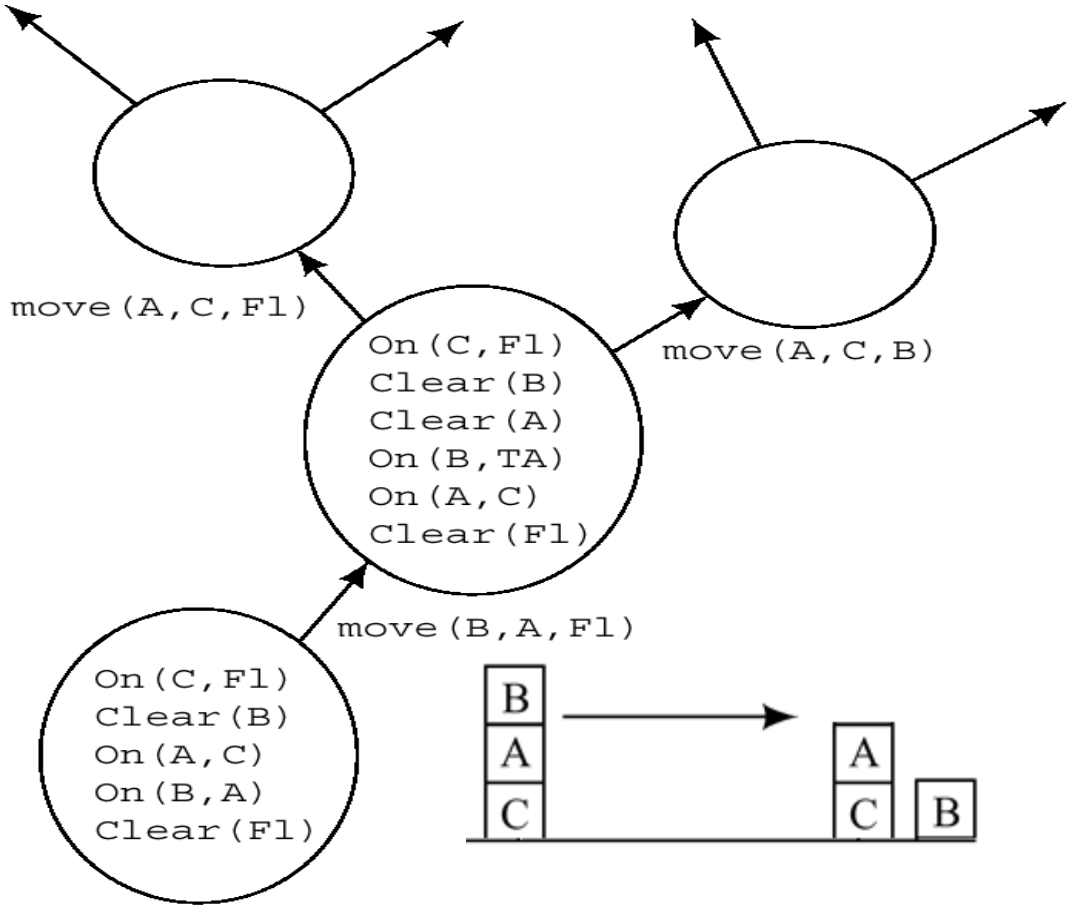


Figure 11.5 Two approaches to searching for a plan. (a) Forward (progression) state-space search, starting in the initial state and using the problem's actions to search forward for the goal state. (b) Backward (regression) state-space search: a belief-state search (see page 84) starting at the goal state(s) and using the inverse of the actions to search backward for the initial state.

Forward Search Methods:
can use A* with some h and g



But, we need good heuristics

Backward: Recursive STRIPS

- **Forward** search with islands:
  - Achieve one subgoal at a time. Achieve a new conjunct without ever violating already achieved conjuncts or maybe temporarily violating previous subgoals.
- **General Problem Solver** (GPS) by Newell Shaw and Simon (1959) uses **Means-Ends** analysis.
- Each subgoal is achieved via a matched rule, then its preconditions are subgoals and so on. This leads to a planner called STRIPS(gamma) when gamma is a goal formula.

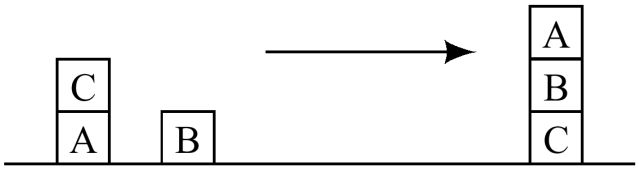
Recursive STRIPS algorithm

- Given a goal stack:
- 1. Consider the top goal
- 2. Find a sequence of actions satisfying the goal from the current state and apply them.
- 3. The next goal is considered from the new state.
- 4. Temination: stack empty
- 5. Check goals again.

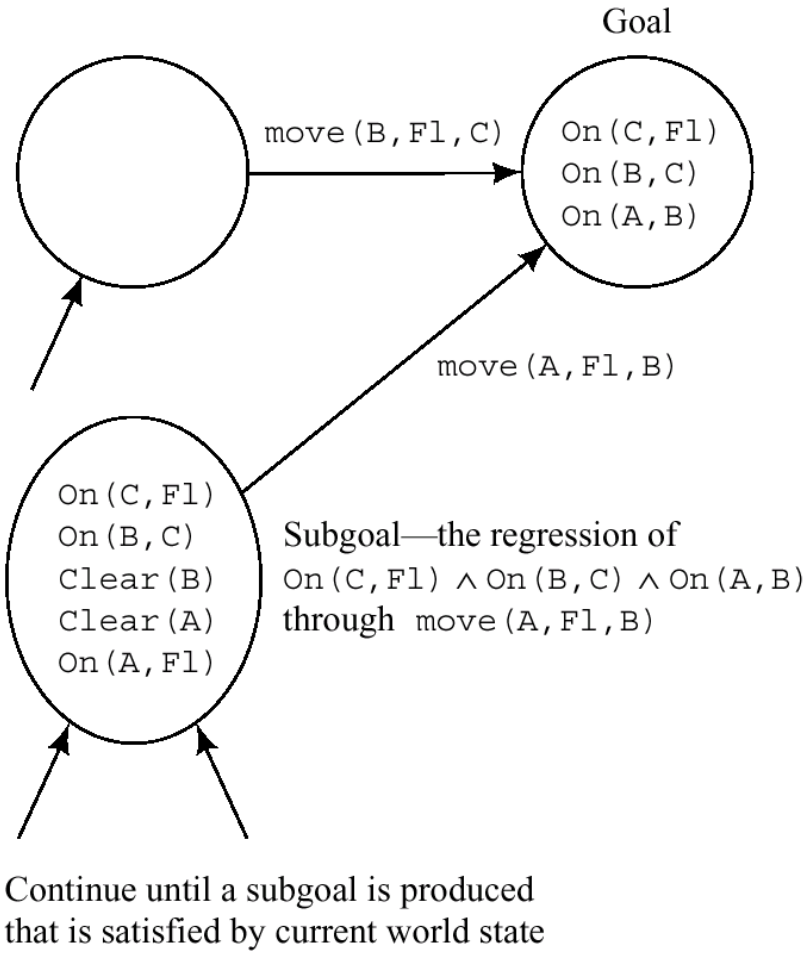
The Sussman anomaly

- RSTRIPS cannot achieve shortest plan
- Two possible orderings of subgoals:
  - [On(A,B) and On(B,C)] or [On(B,C) and On(A,B)]

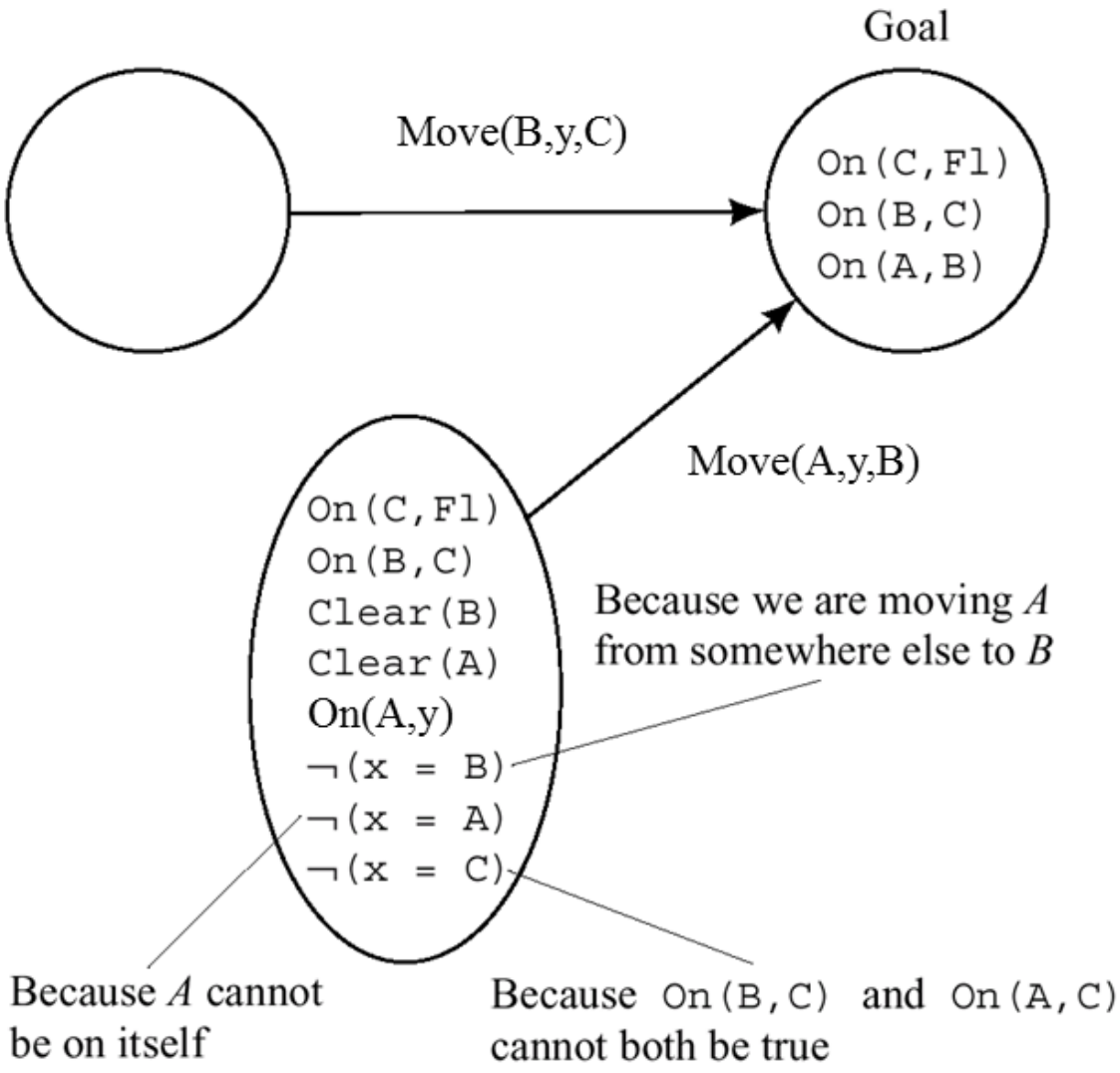
Backward search methods



- Regressing a ground operator



Regressing an ungrounded operator



Move(x,y,z)

PC: On(x,y), Clear(x), Clear(z)

EL: ¬Clear(z), ¬On(x,y), On(x,z), Clear(y), Clear(Fl)

⇓

Unify : On(x,z) and On(A,B)

⇓

Move(A,y,B)

PC: On(A,y), Clear(A), Clear(B)

EL: ¬Clear(B), ¬On(A,y), On(A,B), Clear(y), Clear(Fl)

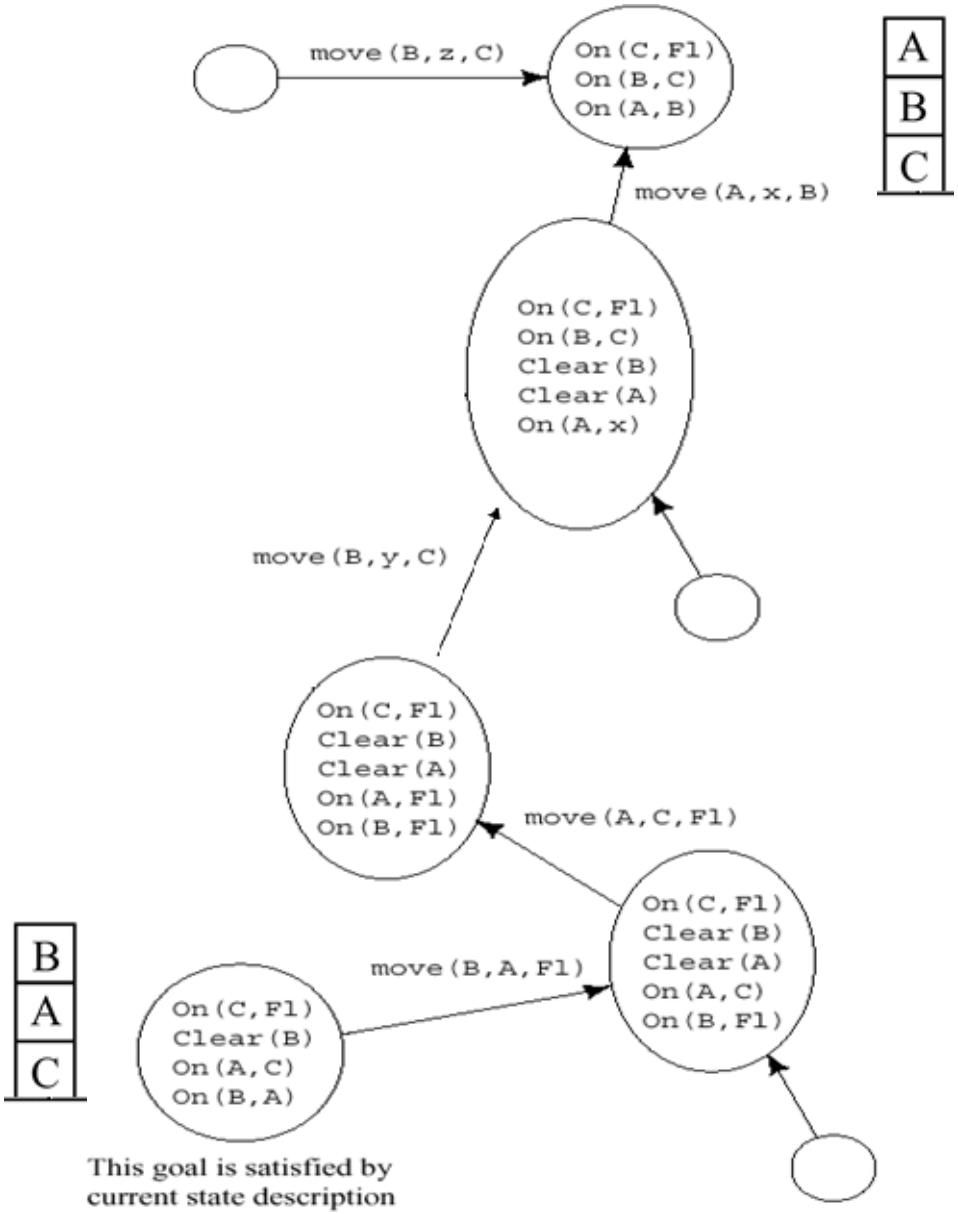
⇓

Check EL does not interfere with goal

⇓

Replace On(A,B) with PC list

Example of Backward Search



Heuristics for planning

- Use relax problem idea to get lower bounds on least number of actions to the goal.
  - Remove all or some preconditions
- Sub-goal independence: the cost of solving a set of subgoals equals the sum cost of solving each one independently, or max cost
  - Can be pessimistic (interacting sub-plans) – not admissible, but can still be useful
  - Can be optimistic (negative effects)
- Simple: number of unsatisfied sub-goals.
- Various ideas related to removing negative effects or positive effects.

More on heuristic generation

- Ignore pre-conditions
  - Still NP-hard for optimal solution
- Ignore delete list: allow making monotone progress toward the goal.
  - Still NP-hard for optimal solution, but hill-climbing algorithms using an approximate solution that is polynomial. (example, 15 puzzle)
- Abstraction: Combines many states into a single one: Pattern databases
- FF : Fast-forward planner (Hoffman 2005), a forward state-space planner with delete-list based heuristic

Partial order planning

- Least commitment planning
- Nonlinear planning
- Search in the space of partial plans
- A state is a partial incomplete partially ordered plan
- Operators transform plans to other plans by:
  - Adding steps
  - Reordering
  - Grounding variables
- SNLP: Systematic Nonlinear Planning (McAllester and Rosenblitt 1991)
- NONLIN (Tate 1977)

A partial order plan for putting shoes and socks

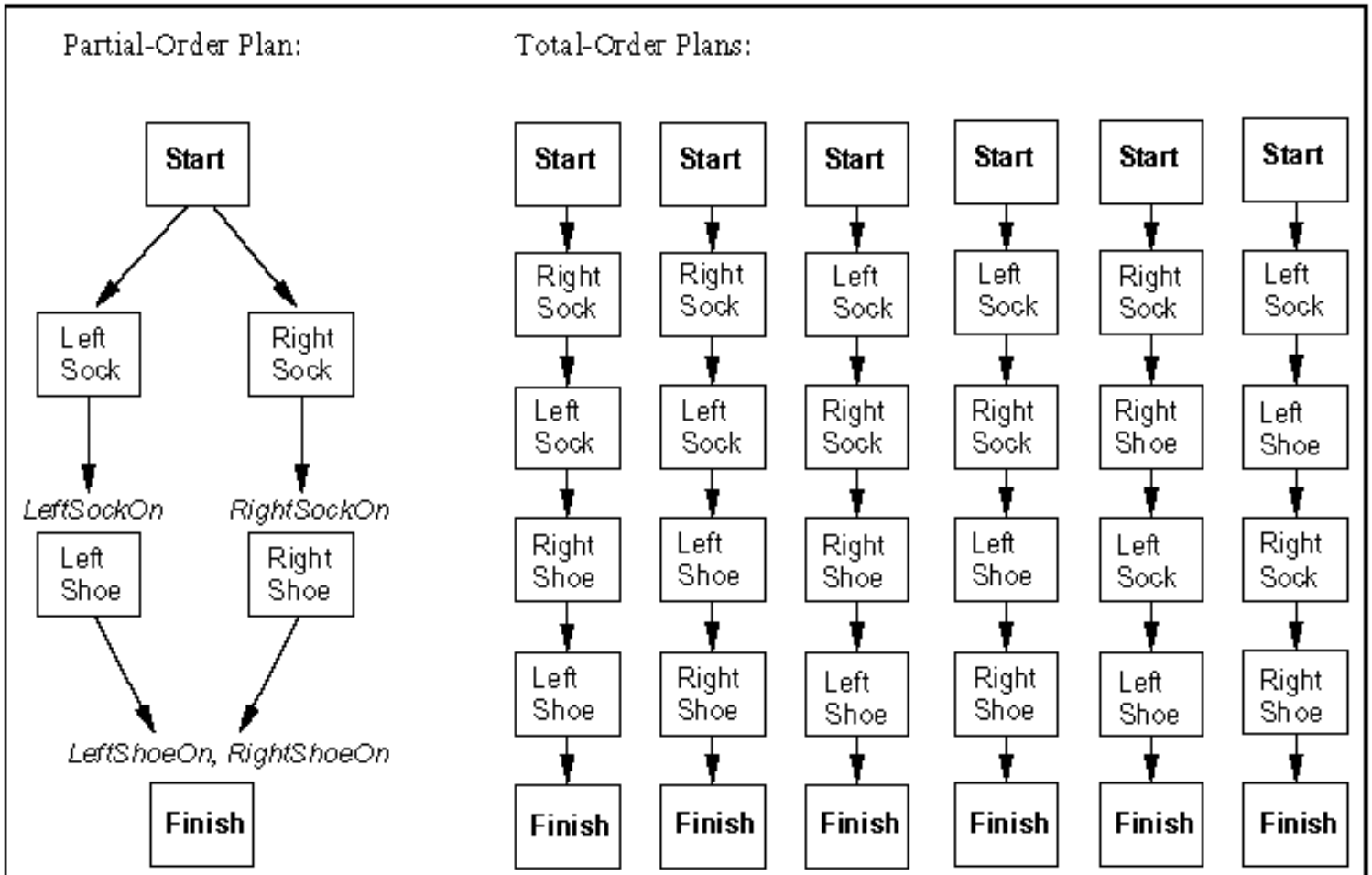


Figure 11.6 A partial-order plan for putting on shoes and socks, and the six corresponding linearizations into total-order plans.

Planning Graphs

- A planning graph consists of a sequence of levels that correspond to time-steps in the plan
- Level 0 is the initial state.
- Each level contains a set of literals and a set of actions
- Literals are those that could be true at the time step.
- Actions are those that their preconditions could be satisfied at the time step.
- Works only for propositional planning.

Example: have cake and eat it too

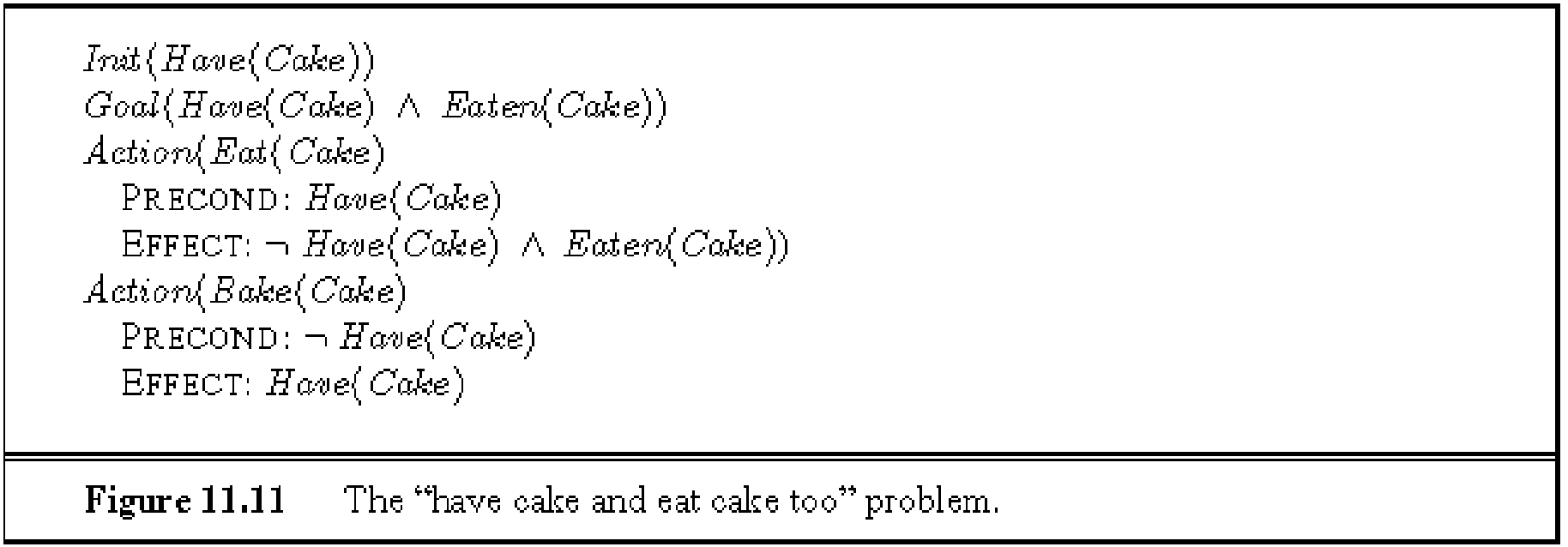


Figure 11.11 The "have cake and eat cake too" problem.

The Planning graphs for “have/eat cake”

- Persistence actions: Represent “inactions” by boxes: frame axiom
- Mutual exclusions (mutex) are represented between literals and actions.
- S_i represents multiple states
- Continue until two levels are identical. The graph levels off.
- The graph records the impossibility of certain choices using mutex links.
- Complexity of graph generation: polynomial in number of literals.

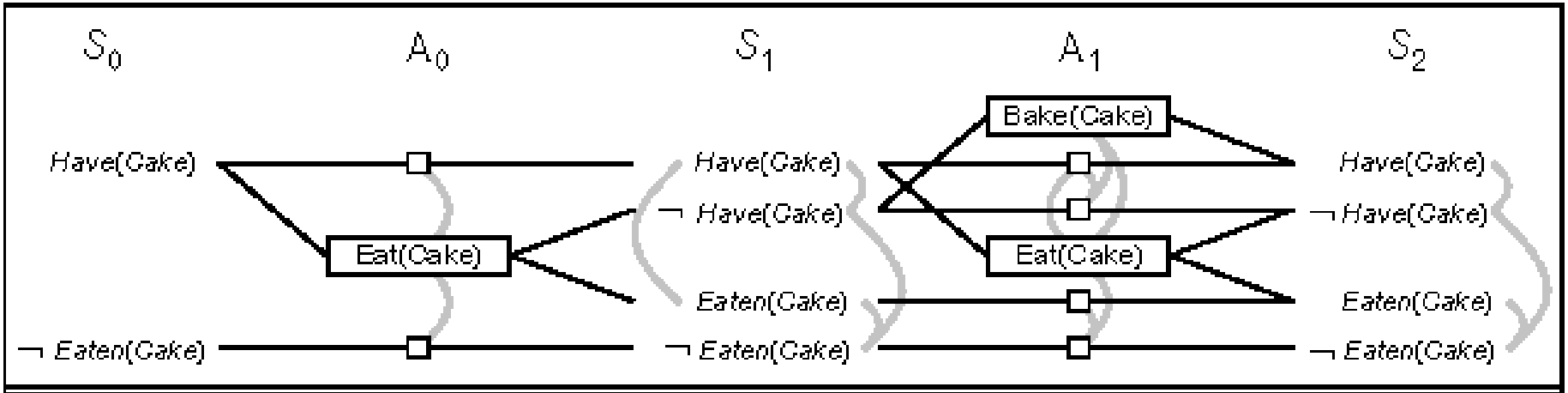


Figure 11.12 The planning graph for the “have cake and eat cake too” problem up to level S_2. Rectangles indicate actions (small squares indicate persistence actions) and straight lines indicate preconditions and effects. Mutex links are shown as curved gray lines.

Defining Mutex relations

- A mutex relation holds between two actions on the same level iff any of the following holds:
  - **Inconsistency effect:** one action negates the effect of another. Example “eat cake and persistence of have cake”
  - **Interference:** One of the effect of one action is the negation of the precondition of the other. Example: eat cake and persistence of Have cake
  - **Competing needs:** one of the preconditions of one action is mutually exclusive with a precondition of another. Example: Bake(cake) and Eat(Cake).
  - **A mutex relation holds between 2 literals** at the same level iff one is the negation of the other or if each possible pair of actions that can achieve the 2 literals is mutually exclusive.

Properties of planning graphs; termination

- Literals increase monotonically
  - Once a literal is in a level it will persist to the next level
- Actions increase monotonically
  - Since the precondition of an action was satisfied at a level and literals persist the action's precond will be satisfied from now on
- Mutexes decrease monotonically:
  - If two actions are mutex at level S_i, they will be mutex at all previous levels at which they both appear
- Because literals increase and mutex decrease it is guaranteed that at some point planning graph will "level off"
- If a set of goals does not appear conflict-free, no valid plan exists
- Can be built in polynomial time

Planning graphs for heuristic estimation

- Estimate the cost of achieving a goal by the level in the planning graph where it appears.
- To estimate the cost of a conjunction of goals use one of the following:
- Max-level: take the maximum level of any goal (admissible)
- Sum-cost: Take the sum of levels (inadmissible)
- Set-level: find the level where they all appear without Mutex (admissible). Dominates max-level

- Graph plans are relaxation of the problem. Representing more than pair-wise mutex is not cost-effective

The GraphPlan algorithm

```
function GRAPHPLAN(problem) returns solution or failure

    graph ← INITIAL-PLANNING-GRAPH(problem)
    goals ← GOALS[problem]
    loop do
        if goals all non-mutex in last level of graph then do
            solution ← EXTRACT-SOLUTION(graph, goals, LENGTH(graph))
            if solution ≠ failure then return solution
            else if NO-SOLUTION-POSSIBLE(graph) then return failure
        graph ← EXPAND-GRAPH(graph, problem)
```

Figure 11.13 The GRAPHPLAN algorithm. GRAPHPLAN alternates between a solution extraction step and a graph expansion step. EXTRACT-SOLUTION looks for whether a plan can be found, starting at the end and searching backwards. EXPAND-GRAPH adds the actions for the current level and the state literals for the next level.

Planning graph for spare tire

goal: at(spare,axle)

- S2 has all goals and no mutex so we can try to extract solutions
- Use either CSP algorithm with actions as variables
- Or search backwards

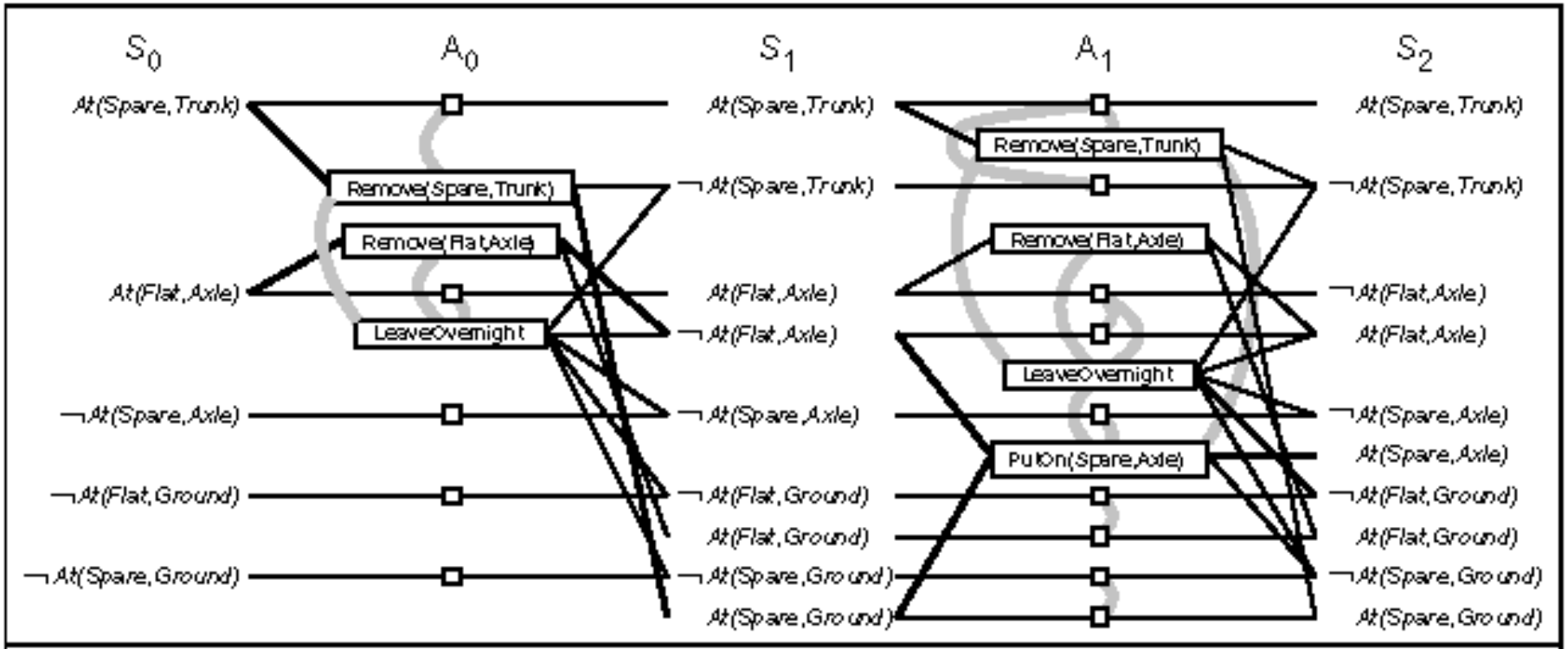


Figure 11.14 The planning graph for the spare tire problem after expansion to level S_2. Mutex links are shown as gray lines. Only some representative mutexes are shown, because the graph would be too cluttered if we showed them all. The solution is indicated by bold lines and outlines.

Search planning-graph backwards with heuristics

- How to choose an action during backwards search:
  - Use greedy algorithm based on the level cost of the literals.
- For any set of goals:
- 1. Pick first the literal with the highest level cost.
- 2. To achieve the literal, choose the action with the easiest preconditions first (based on sum or max level of precond literals).

Other classical planning approaches

- The most effective approached to planning currently are:
  - Translating to Boolean Satisfiability
  - Forward state-space search with carefully crafted heuristics
  - Search using planning graphs (covered already)

Planning as Satisfiability

- Express planning as a set of propositions.
- Fix length of plan at T.
- A propositional variable for each propositions at each time step
  - On(A,B)_0, ON(B,C)_0, etc.
- A propositional variable for each action at each time step t
  - $(a_{1t} \vee a_{2t} \vee \ldots \vee a_{nt})$, also $\neg(a_{it} \,\&\, a_{jt})$
- Action axioms :
  - $a_{it} \Rightarrow PC(a_i)_t \,\&\, EL(a_i)_{t+1}$
- Successor-state axioms need to be expressed for each action (like in the situation calculus but propositional)
  - If a_{it} is true, then either it was true before (at time t-1) and nothing changed it, or some action caused it
- Goal conditions: the goal conjuncts at time T.
- Unknown propositions are not stated.
- Propositions known not to be true are stated negatively.

Planning with propositional logic (continued)

- We write the formula:
  - Initial state and succesor state axioms and goal
- We search for a model to the formula. Those actions that are assigned true constitute a plan.
- We can also choose to allow partial order plans and only write exclusions between actions that interfere with each other.

- Planning: start at T=0 and iteratively try to find longer and longer plans.

SATplan algorithm

```
function SATPLAN(problem, T_max) returns solution or failure
  inputs: problem, a planning problem
          T_max, an upper limit for plan length

  for T = 0 to T_max do
    cnf, mapping ← TRANSLATE-TO-SAT(problem, T)
    assignment ← SAT-SOLVER(cnf)
    if assignment is not null then
      return EXTRACT-SOLUTION(assignment, mapping)
  return failure
```

Figure 11.15 The SATPLAN algorithm. The planning problem is translated into a CNF sentence in which the goal is asserted to hold at a fixed time step T and axioms are included for each time step up to T. (Details of the translation are given in the text.) If the satisfiability algorithm finds a model, then a plan is extracted by looking at those proposition symbols that refer to actions and are assigned *true* in the model. If no model exists, then the process is repeated with the goal moved one step later.

Complexity of SATplan

- The total number of action symbols is:
  - |T|x|Act|x|O|^p
  - O = number of objects, p is scope of atoms.
- Number of clauses is higher.
- Example: 10 time steps, 12 planes, 30 airports, the complete action exclusion axiom has 583 million clauses.

The flashlight problem (from Steve Lavelle)

- **Figure 2.18:** Three operators for the flashlight problem. Note that an operator can be expressed with variable argument(s) for which different instances could be substituted.
- http://planning.cs.uiuc.edu/node59.html#for:strips
- Here is a satplan for flashlight Battery
- http://planning.cs.uiuc.edu/node68.html

Summary: Planning

- STRIPS Planning
- Forward and backward planning
- Partial order planning
- Situation Calculus
- GraphPlan
- SATplan
- Readings: RN chapter 10