Set 9: Planning
Classical Planning Systems
Chapter 10 R&N

ICS 271 Fall 2015

Outline: Planning

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

What is planning?

- “Planning is a task of finding a sequence of actions that will transfer the initial world into one in which the goal description is true.”
- “The planning can be seen as a sequence of actions generator which are restricted by constraints describing the limitations on the world under view.”
- “Planning as the process of devising, designing or formulating something to be done, such as the arrangements of the parts of a thing or an action or proceedings to be carried out.”

Setup

- Actions : deterministic/non-deterministic?
- State variables : discreet/continuous?
- Current state : observable?
- Initial state : known?
- Actions : duration?
- Actions : 1 at a time?
- Objective : reach a goal? maximize utility/reward?
- Agent : 1 or more? Cooperative/competitive?
- Environment : Known/unknown, static?

Setup

- Classical planning:
  - Actions : deterministic
  - States : fully observable, initial state known
  - Environment : known and static
  - Objective : reach a goal state
- Games
  - Agents : 2 (or more) competing
  - Objective : maximize utility
- Conformant planning:
  - Actions : non-deterministic
  - States : not observable, initial state unknown
  - Objective : maximize probability of reaching the goal
- Markov decision process (MDP):
  - Actions : non-deterministic with probabilities known
  - States : fully observable
  - Objective : maximize reward

Planning vs Scheduling

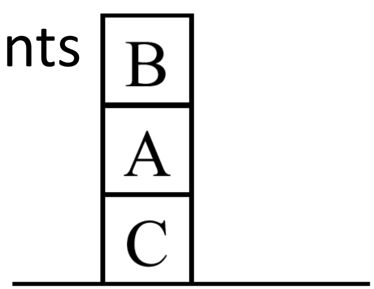
- Objective :
  - find a sequence of actions
  - find an allocation of jobs to resources
- Solution
  - Plan length unknown
  - Number of jobs to schedule known
- Complexity
  - PSPACE (planning)
  - NP-hard (scheduling)

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 knowledge base.
- 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,Floor): move block A from block B to Floor
- move(x,y,z) - action schema
- **do**: A function constant, **do** denotes a function that maps actions and states into states
  - $do(\alpha, \sigma) \to \sigma_1$

    action

    state

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,Floor) 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:

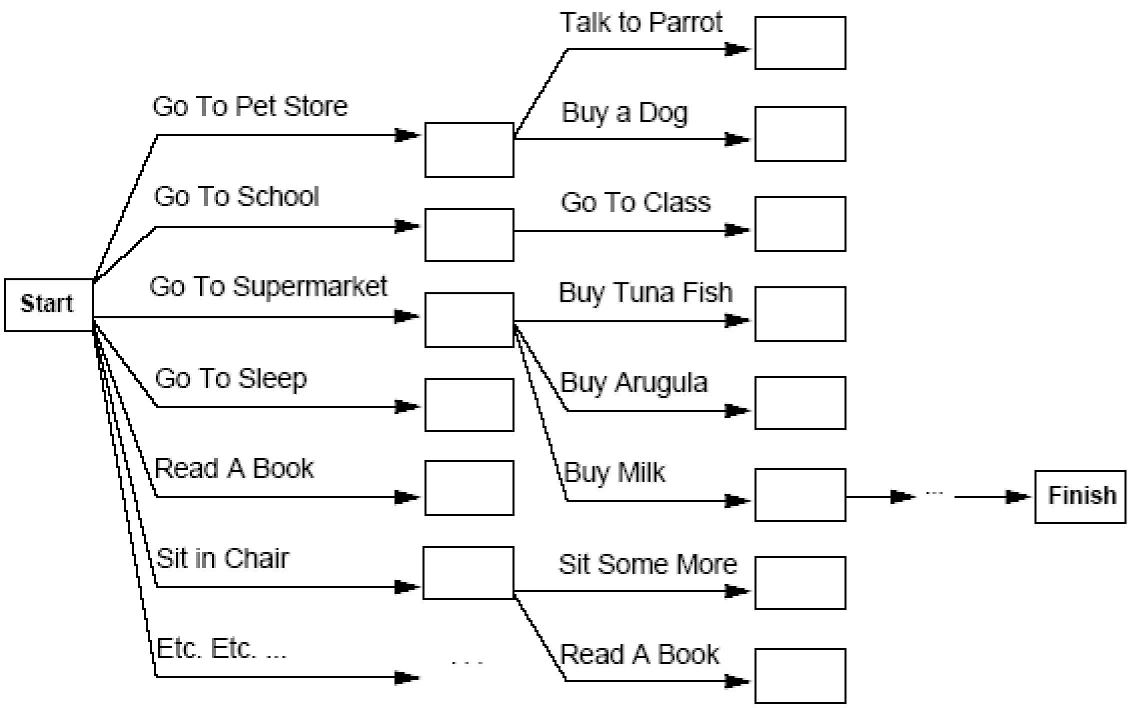
$$(\neg On(x,y,s) \wedge [(x \neq u) \vee (y \neq z)]) \rightarrow \neg On(x,y,do(move(u,v,z),s)$$

STRIPS Planning systems
PDDL: Planning Domain Definition Language

Search vs. planning

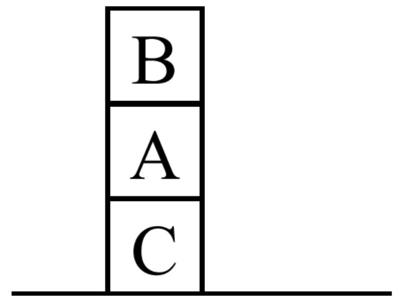
Consider the task *get milk, bananas, and a cordless drill*

Standard search algorithms seem to fail miserably:



After-the-fact heuristic/goal test inadequate

STRIPS: describing goals and state




```
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)
- **State descriptions**: conjunctions of ground functionless atoms
  - Factored representation of states!
- A formula describes a set of world states : On(A,B) $\wedge$ Clear(A)
- Lifted version (schema): On(x,B) $\wedge$ Clear(x)
- Initial state is a conjunction of ground atoms
- Planning search for a formula satisfying a goal description
  - Goal wff: $\exists x[g(x) \wedge f(y)]$
  - Given a goal wff, the search algorithm looks for a sequence of **actions** that transforms initial state into a state description that entails the goal wff.

STRIPS: description of actions

- A STRIPS operator (action) 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)*
- Lifting from prop logic level of representation to FOL level of representation
- 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

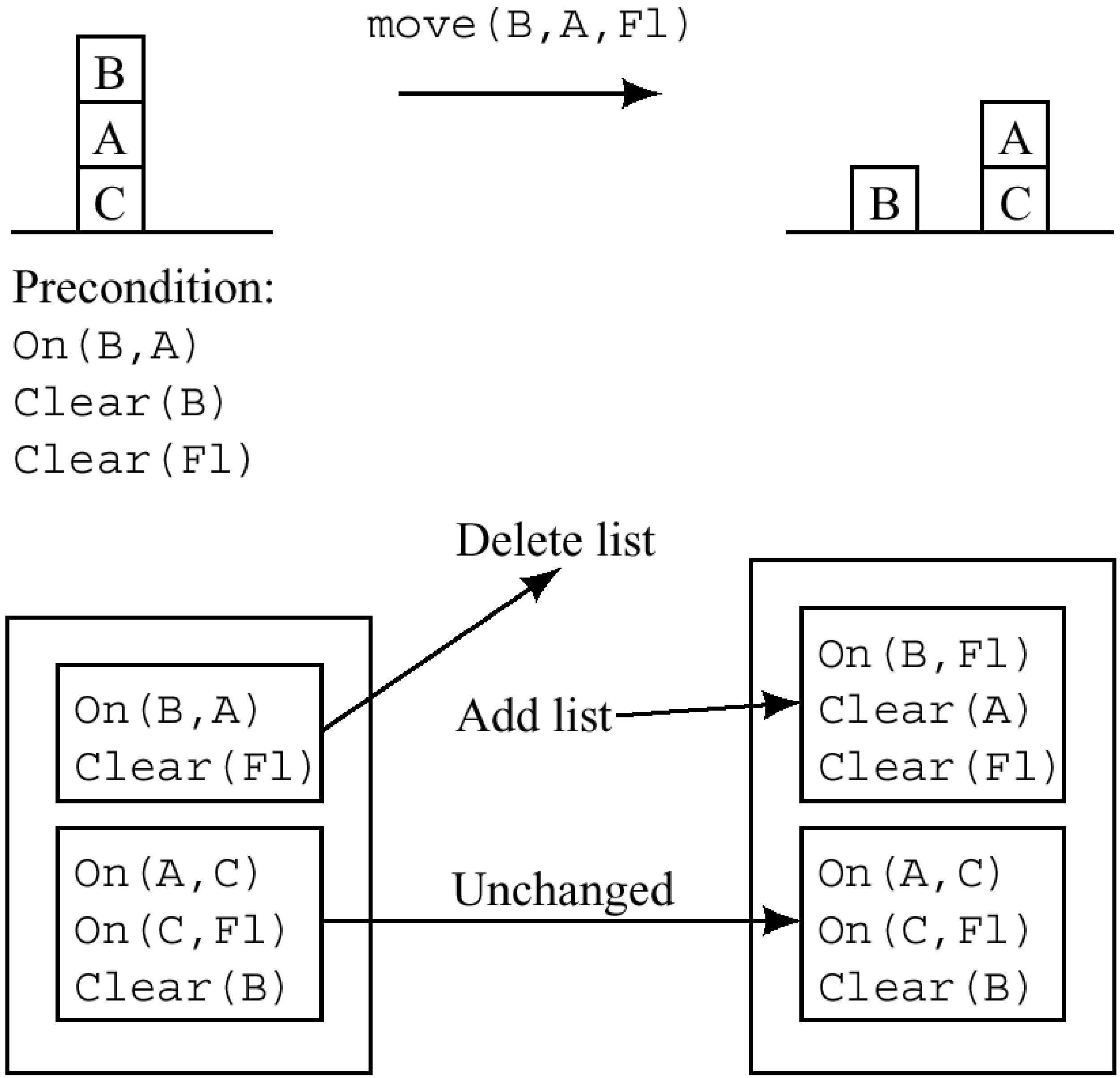
ACTION: $Buy(x)$
PRECONDITION: $At(p), Sells(p, x)$
EFFECT: $Have(x)$

[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



PDDL vs STRIPS

- A language that yields a search problem : actions translate into operators in search space
- *PDDL is a slight generalization of STRIP language*
- A state is
  - a set of positive ground literals (STRIPS)
  - a set of ground literals (PDDL)
- Closed world assumption : fluents that are not mentioned are false (STRIPS).
- If a literals is not mentioned, it is unknown (PDDL).
- Action schema:

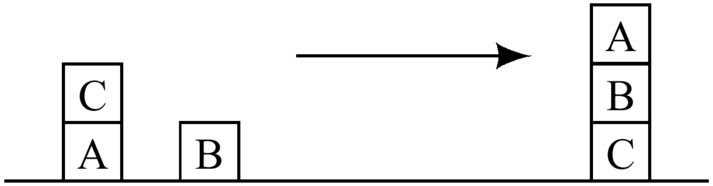
Action(Fly(p,from,to)):

Precond: $At(p,from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$

Effect: $\neg At(p,from) \wedge At(p,to)$

- *The schema consists of precondition and effect lists :*
  - *Only positive preconditions (STRIPS)*
  - *Positive or negative preconditions (PDDL)*
- *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 atoms) and a goal: conjunction of atoms, some not grounded (At(p,SFO), Plane(p))*

The block world



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

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

$Init(At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$
$\quad \wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$
$\quad \wedge Airport(JFK) \wedge Airport(SFO))$
$Goal(At(C_1, JFK) \wedge At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg At(c, a) \wedge In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c, a) \wedge \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg At(p, from) \wedge At(p, to))$

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

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

Figure 11.3 The simple spare tire problem.

Complexity of classical planning

- Tasks
  - PlanSAT = decide if plan exists
  - Bounded PlanSAT = decide if plan of given length exists
- (Bounded) PlanSAT decidable but PSPACE-hard
- Disallow neg effects, (Bounded) PlanSAT NP-hard
- Disallow neg preconditions, PlanSAT in P but finding optimal (shortest) plan still NP-hard

Recursive STRIPS

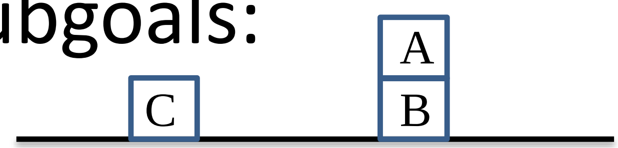
- **STIRPS** algorithm :
  - Divide-and-Conquer forward search with islands
  - Achieve one subgoal at a time : achieve a new goal literal without ever violating already achieved goal literals or maybe temporarily violating previous subgoals.
- Motivated by **General Problem Solver** (GPS) by Newell Shaw and Simon (1959) - **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

- Algorithm maintains a set of goals
  - Start with all problem instance goals
  - At each iterations, take and satisfy one goal
- Algorithm :
  1. Take a goal from goal set
  2. Find a sequence of actions satisfying the goal from the current state, apply the actions, resulting in a new state.
  3. If stack empty, then done.
  4. Otherwise, the next goal is considered from the new state.
  5. At the end, check goals again.

The Sussman anomaly

- RSTRIPS cannot find a valid plan
- Two possible orderings of subgoals:
  - On(A,B) and On(B,C) or



  - On(B,C) and On(A,B)

- Non-interleaved planning does not work if goals are dependent

Algorithms for Planning as State-space Search

- Forward (progression) state-space search
  - Search with applicable actions
- Backward (regression) state-space search
  - Search with relevant actions
- Heuristic search
- Planning graphs

Planning forward and backward

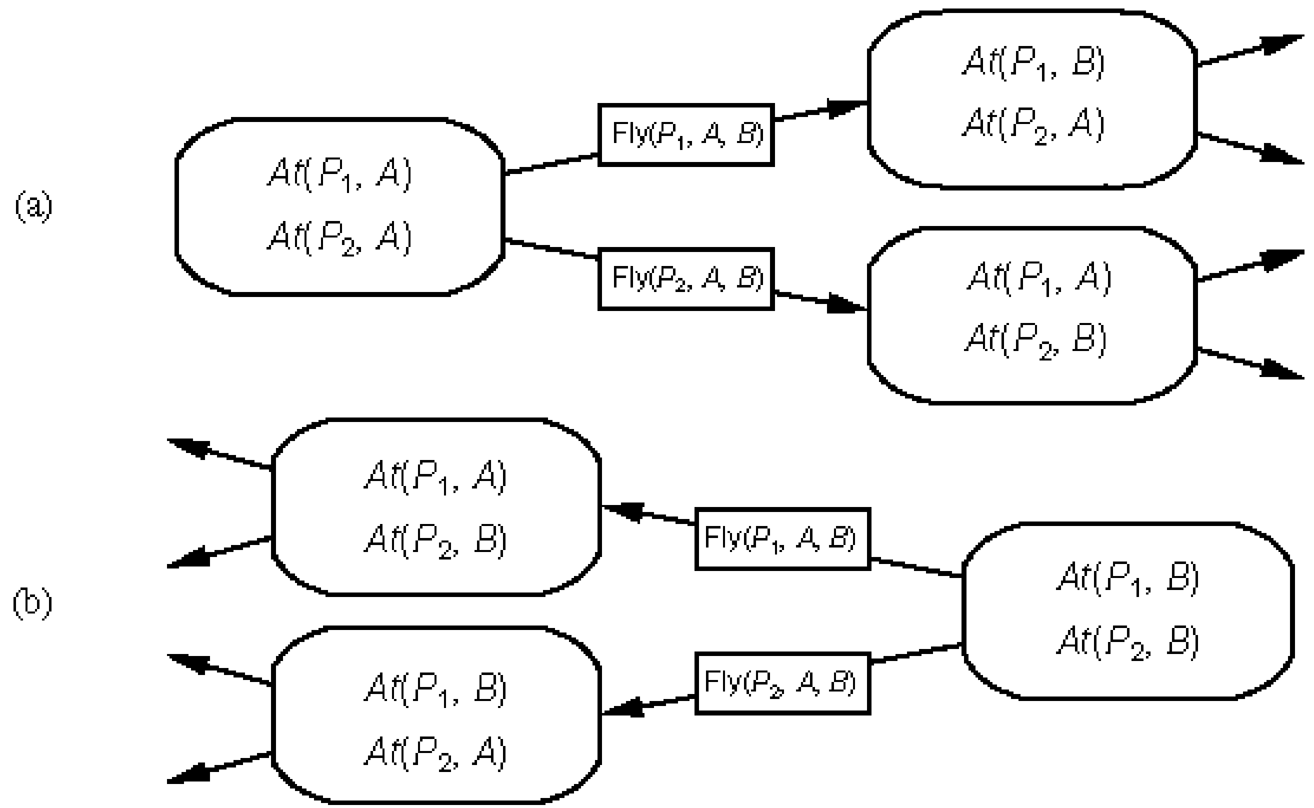
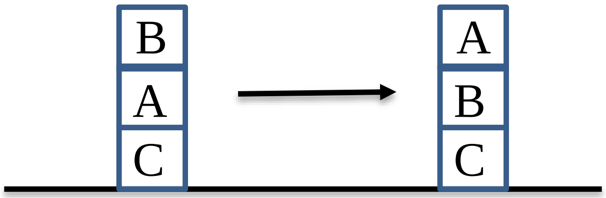
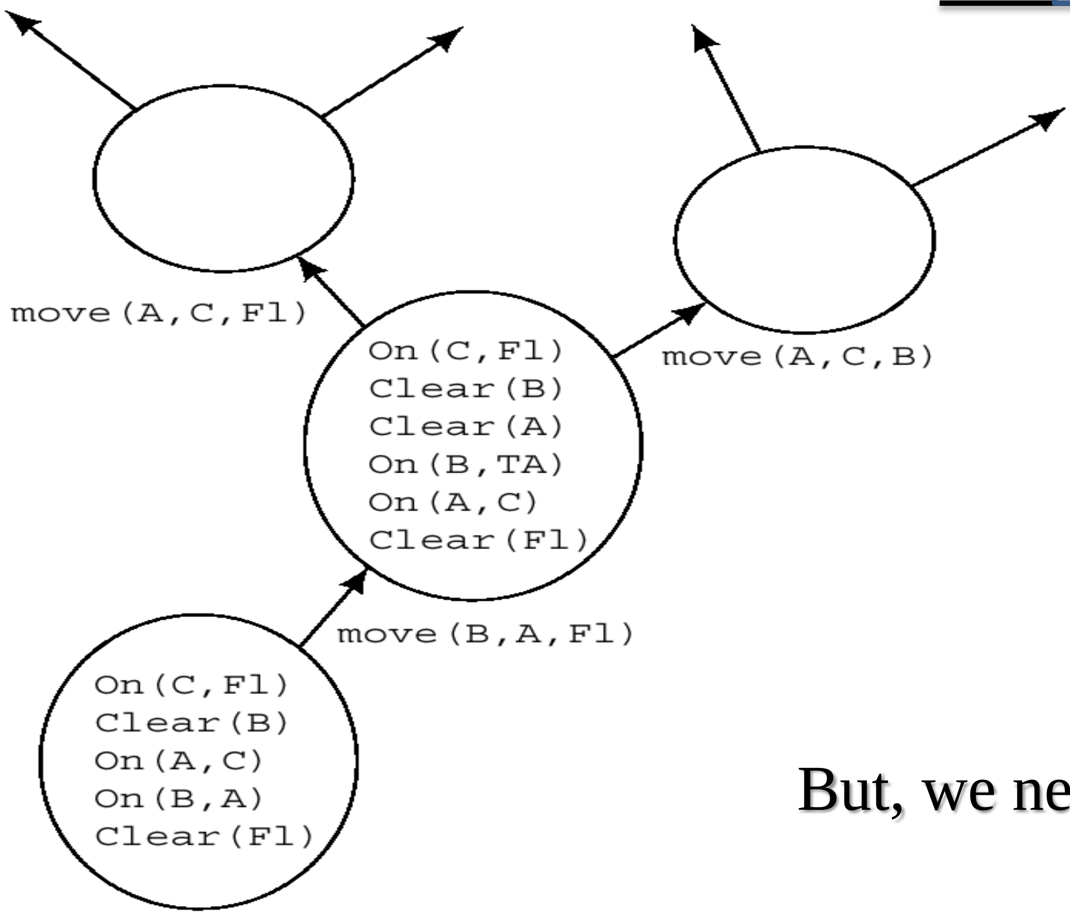


Figure 11.5 Two approaches to searching for a plan. (a) Forward (progression) state-space search, starting in the initial state and using the problem's actions to search forward for the goal state. (b) Backward (regression) state-space search: a belief-state search (see page 84) starting at the goal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



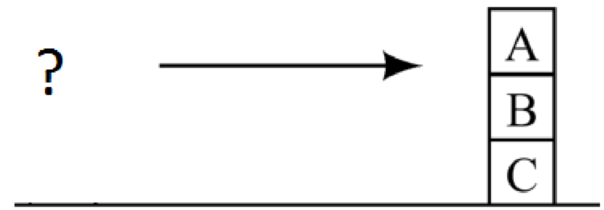


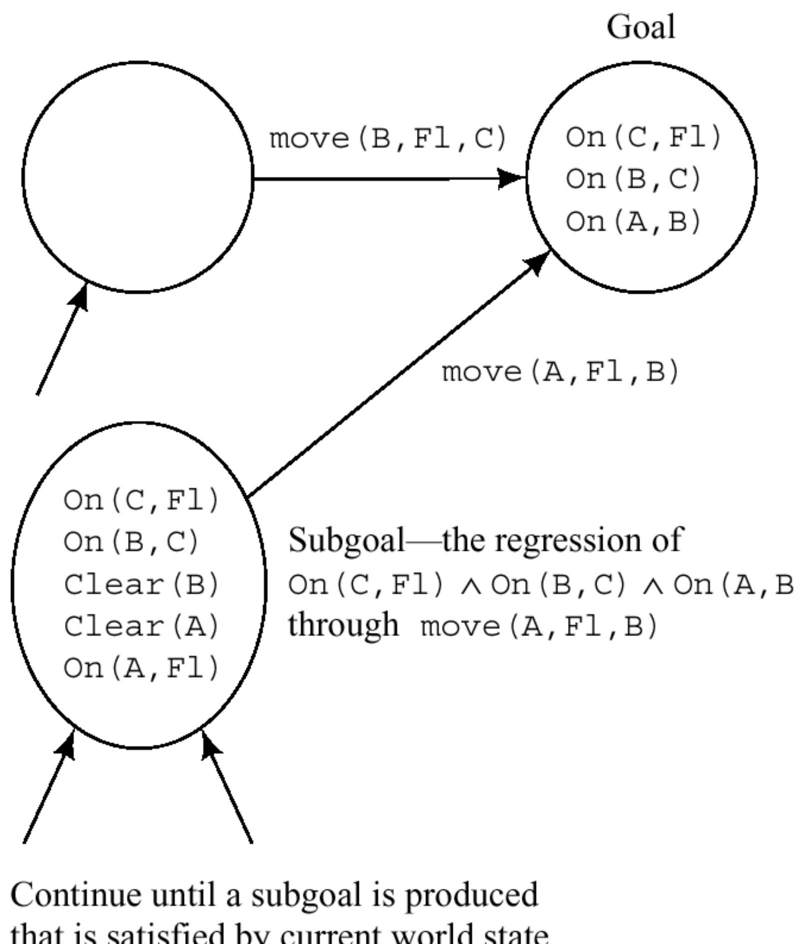
But, we need good heuristics

Backward search methods

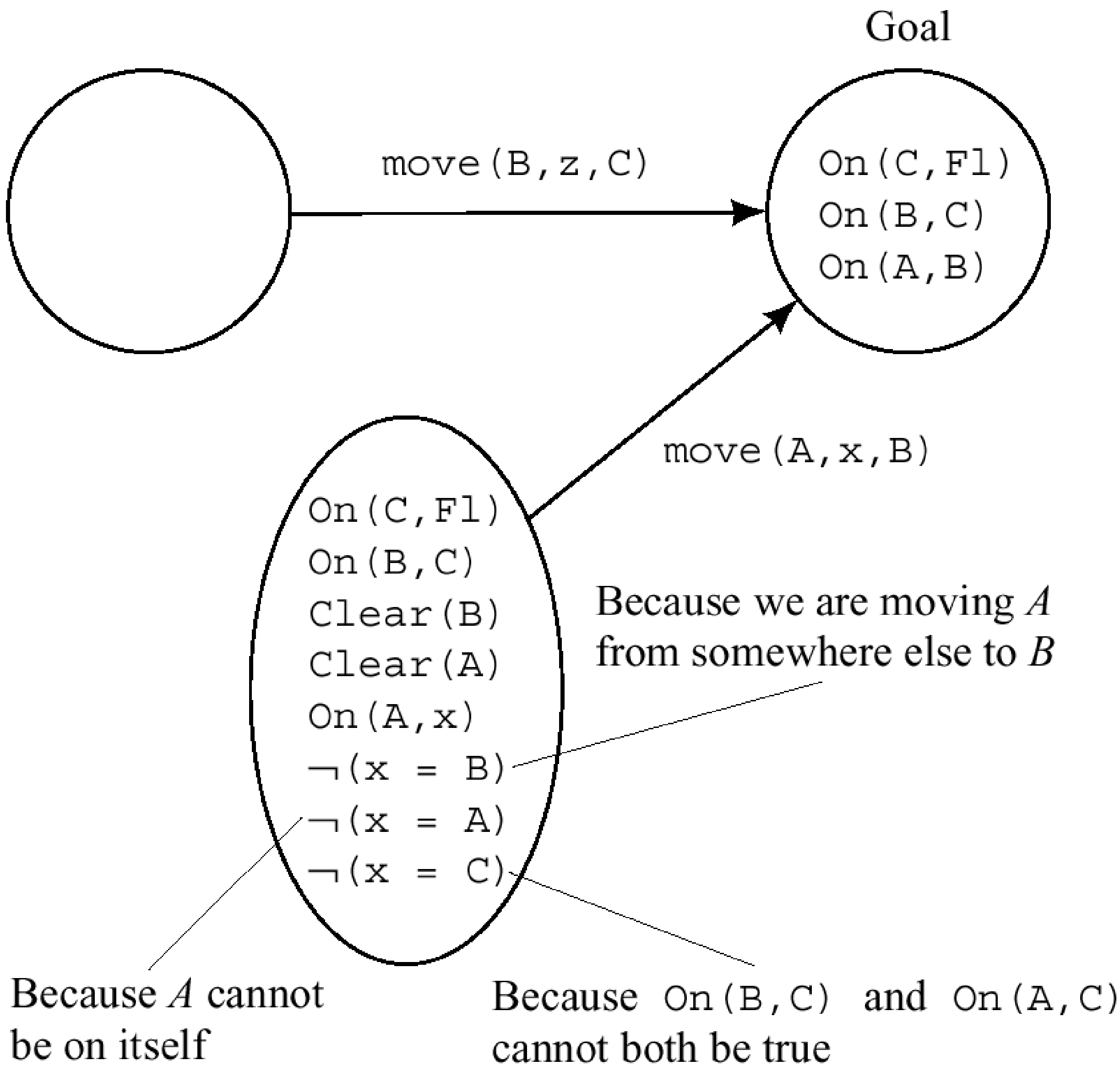
- Regressing a ground operator :

  g' = (g – ADD(a)) ∪ PreCond(a)

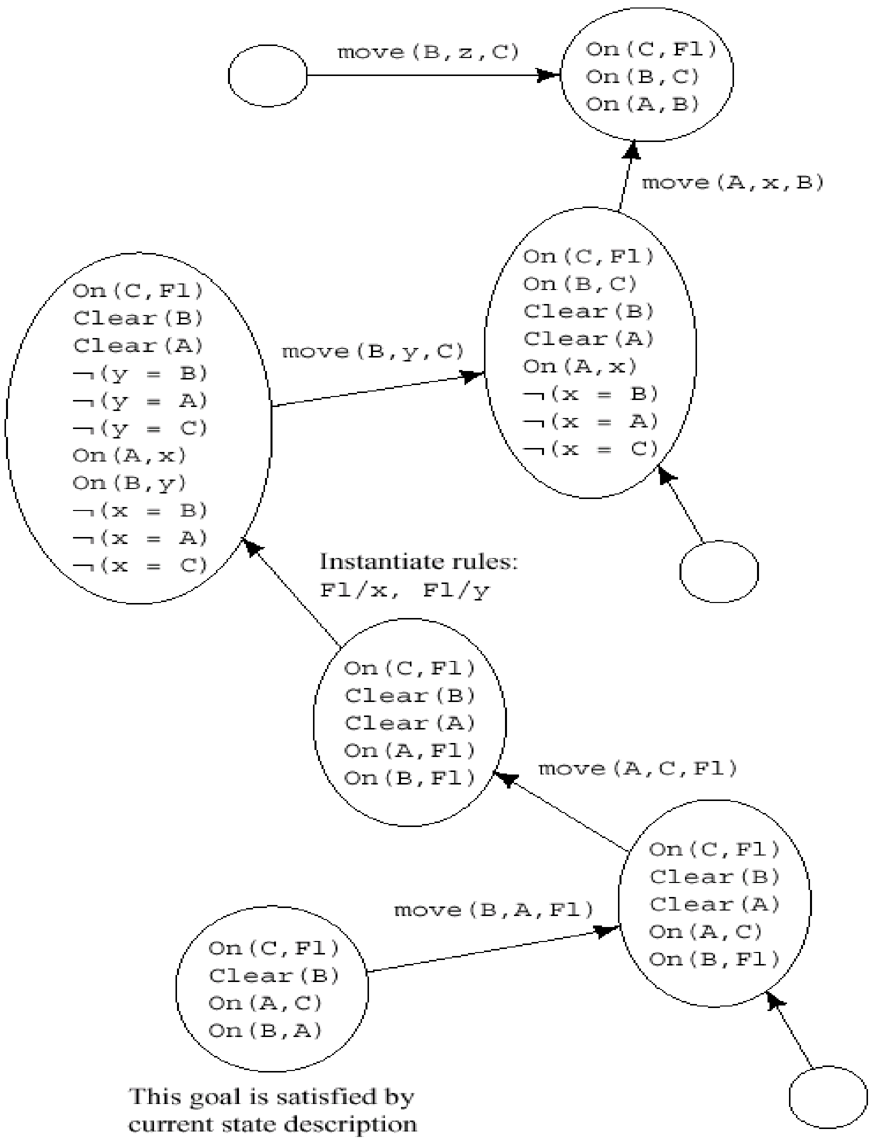




Regressing an action schema



Example of Backward Search



Forward vs Backward planning search

- Forward search space nodes correspond to individual (grounded) states of the plan state-space
- Backward search space nodes correspond to sets of plan state-space states, due to un-instantiated variables
  - because of this, designing good heuristics is hard(er)
  - however, it has smaller branching factor than FS

- Forward search only feasible if good heuristics available

Heuristics for planning

- Use relax problem idea to get lower bounds on least number of actions to the goal
  - Add edges to the plan state-space graph
    - E.g. remove all or some preconditions
  - State abstraction (combining states)
- Sub-goal independence: compute the cost of solving each subgoal in isolation, and combine the costs, e.g. the sum of costs of solving each, or max cost
  - Can be pessimistic (interacting sub-plans)
  - 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 (example, 15 puzzle) : still hard, approximation easy but may not be admissible
- Ignore delete list: allow making monotone progress toward the goal.
  - Still NP-hard for optimal solution, but hill-climbing algorithms find an approximate solution in polynomial time that is admissible
- Abstraction: Combines many states into a single one: E.g. ignore some fluents, pattern databases
- FF : Fast-forward planner (Hoffman 2005), a forward state-space planner with
  - greedy search
  - Ignore-delete-list based heuristic
  - using planning graph to compute heuristic value

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

$Init(Have(Cake))$
$Goal(Have(Cake) \land Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg Have(Cake) \land Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg Have(Cake)$
  EFFECT: $Have(Cake)$

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

The Planning graphs for "have cake",

- Persistence actions: Represent "inactions" by boxes: frame axiom
- Mutual exclusions (mutex) are represented between literals and actions.
- S1 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.

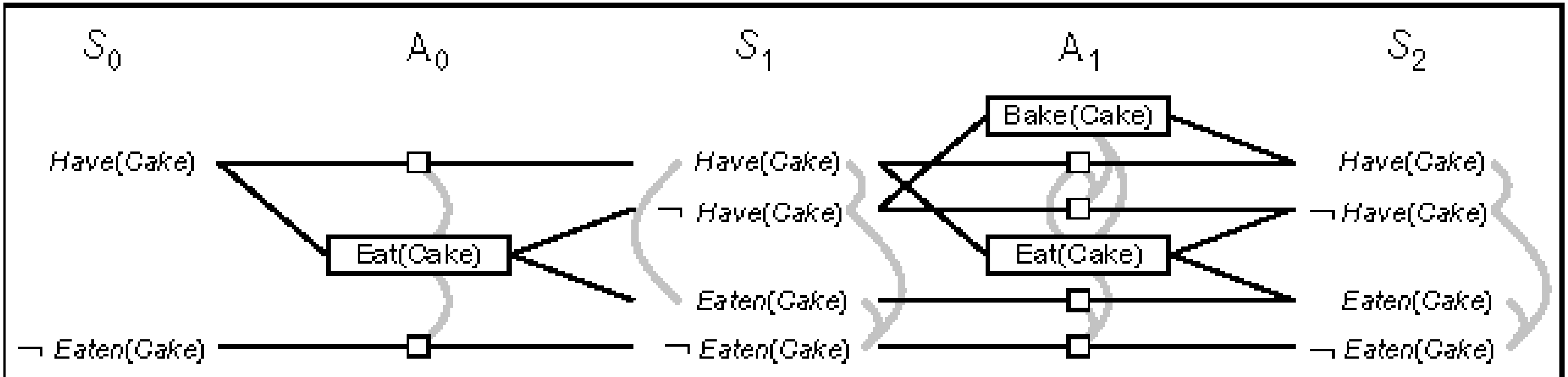


Figure 11.12 The planning graph for the "have cake and eat cake too" problem up to level S_2. Rectangles indicate actions (small squares indicate persistence actions) and straight lines indicate preconditions and effects. Mutex links are shown as curved gray lines.

Defining Mutex relations

- A mutex relation holds between 2 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 precondition 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
  - If two literals are not mutex, they will always be non-mutex later
- Because literals increase and mutex decrease it is guaranteed that we will have a level where $S_i = S_{i-1}$ and $A_i = A_{i-1}$ that is, the planning graph has stabilized

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.

- Note, we don't have to build planning graph to completion to compute heuristic estimates

- Graph plans are an approximation of the problem. Representing more than pair-wise mutex is not cost-effective
  - E.g. On(A,B), On(B,C), On(C,A)

The GraphPlan algorithm

- Start with a set of problem goals G at the last level S
- At each level S_i, select a subset of conflict-free actions A_i for the goals of G_i, such that
  - Goals G_i are covered
  - No 2 actions in A_i are mutex
  - No 2 preconditions of any 2 actions in A_i are mutex
- Preconditions of A_i become goals of S_{i-1}
- Success iff G_0 is subset of initial state

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

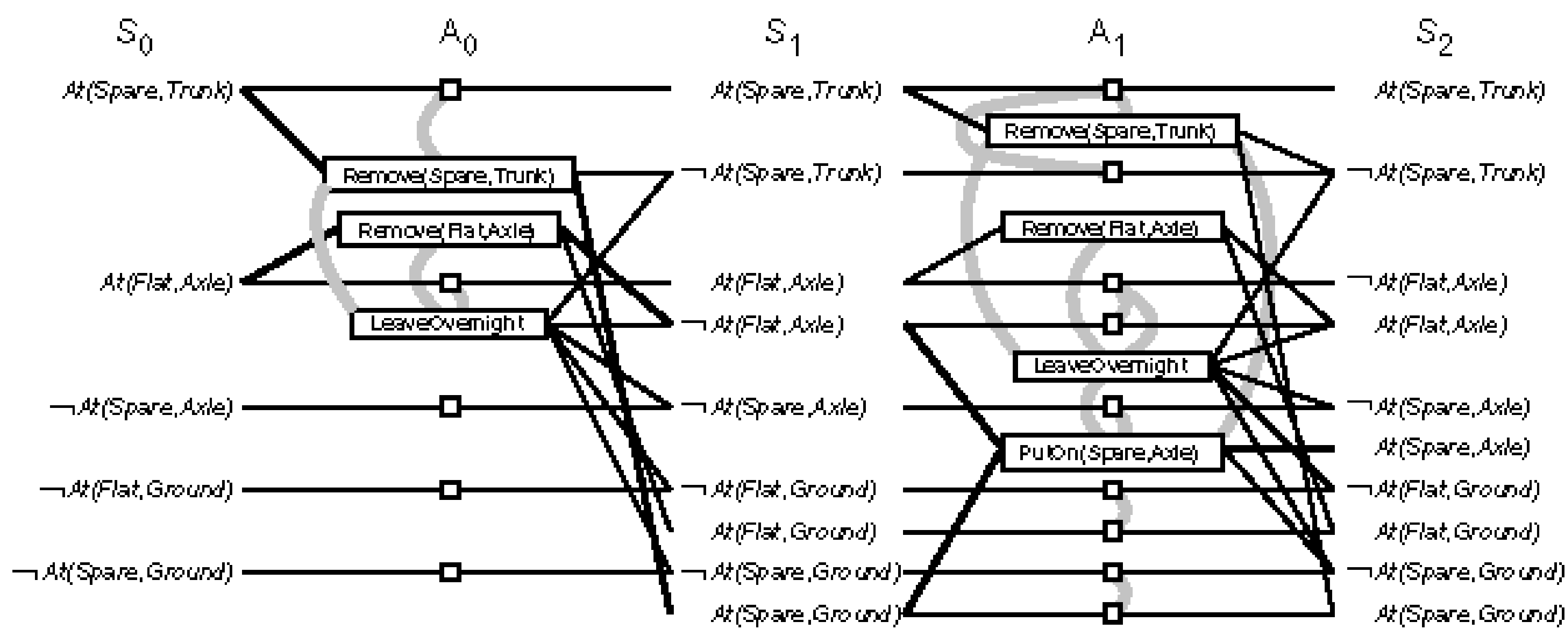


Figure 11.14 The planning graph for the spare tire problem after expansion to level S_2. Mutex links are shown as gray lines. Only some representative mutexes are shown, because the graph would be too cluttered if we showed them all. The solution is indicated by bold lines and outlines.

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.

Searching 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 precondition literals).

Main classical planning approaches

- The most effective approaches to planning currently are:
  - Forward state-space search with carefully crafted heuristics
  - Search using planning graphs (GraphPlan or CSP)
  - Translating to Boolean Satisfiability

Planning as Satisfiability

- Express propositional planning as a set of propositions.
- Index propositions with time steps:
  - On(A,B)_0, ON(B,C)_0
- Goal conditions:
  - the goal conjuncts at time T, T is determined arbitrarily.
- Unknown propositions are not stated.
- Propositions known not to be true are stated negatively.
- Actions: a proposition for each action for each time slot.
- Successor state axioms need to be expressed for each action (like in the situation calculus but it is propositional)

Planning with propositional logic (continued)

- We write the formula:
  - initial state and action effect/precondition axioms and successor state axioms and goal state
- We search for a model to the formula. Those actions that are assigned true constitute a plan.
- To have a single plan we may have a mutual exclusion for all actions in the same time slot.
- We can also choose to allow partial order plans and only write exclusions between actions that interfere with each other.

- Planning: 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

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

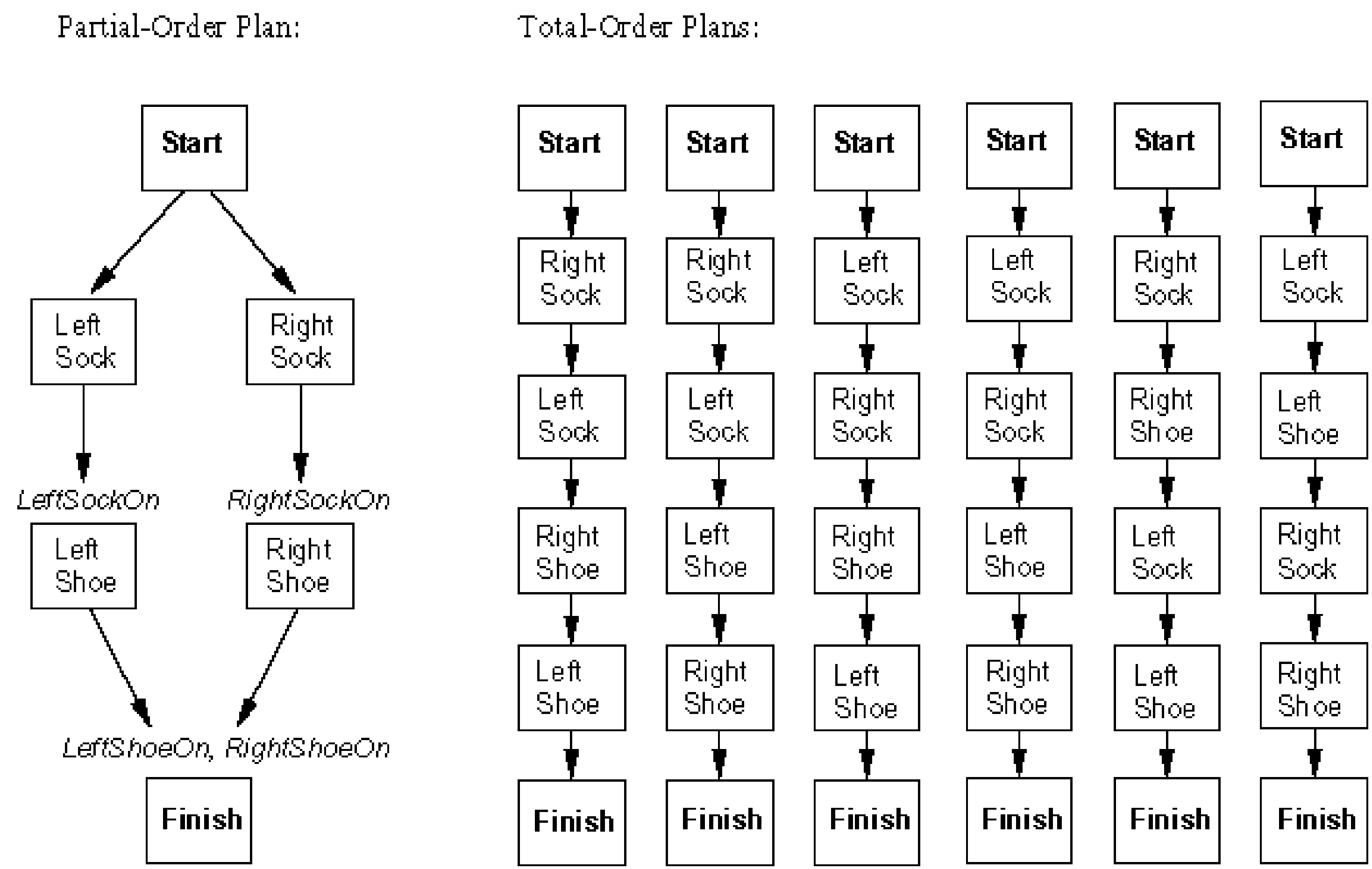


Figure 11.6 A partial-order plan for putting on shoes and socks, and the six corresponding linearizations into total-order plans.

Summary: Planning

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