Planning

Berlin Chen 2003

References:

1. S. Russell and P. Norvig. Artificial Intelligence: A Modern Approach, Chapters 10-12
2. S. Russell's teaching materials

Introduction

- Planning is he task of coming up with a sequence of actions that will achieve a goal
  - Open up action and goal representation to allow selection
  - Divide-and-conquer by subgoaling
  - Relax requirement for sequential construction of solutions

	Search	**Planning**
States	Lisp data structures	Logical sentences
Actions	Lisp code	Preconditions/outcomes
Goal	Lisp code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

  - Algorithms should take advantage of the structure of the logical representation of the problem

Buy(*x*) ↔ *Have*(*x*)

Buy(*ISBN*0137903952) ↔ *Have*(*ISBN*0137903952)

Introduction

- The environments considered first are fully observable, deterministic, finite, static and discrete
  - Called classical planning

- Find a good domain-independent heuristic function ?
  - Goal test as a block box in traditional search-based problem-solving
  - Try to explicitly represent the goal as a conjunction of subgoals
    - A logical representation

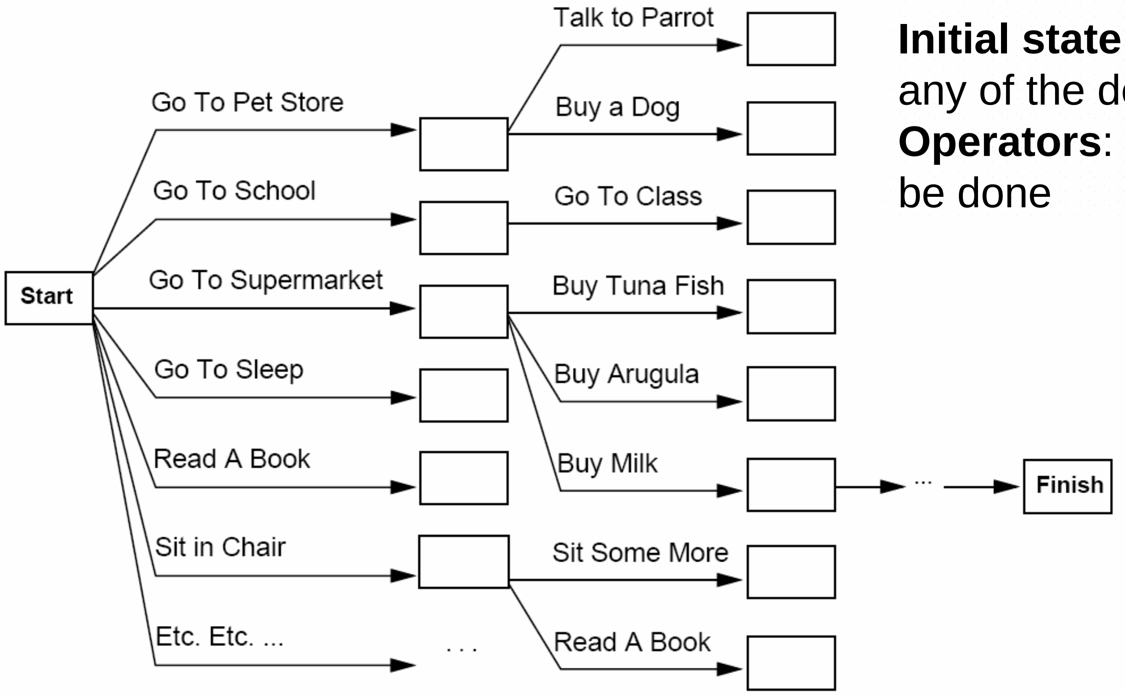
$$Have(A) \wedge Have(B) \wedge Have(C) \wedge Have(D)$$

    - Perfectly decomposable problems are delicious and rare
      - Interactions among subgoals

Example: Problem-solving Agent

- Task Goal
  - To get a quart of milk
  - A bunch of bananas
  - A variable-speed cordless drill

Have(*Milk*) ∧ *Have*(*Bananas*) ∧ *Have*(*Drill*)



Initial state: at home but without any of the desired objects
Operators: all the things can be done

- Often overwhelmed by irrelevant actions

Languages of Planning Problems

- Major specifications of planning problems
  - States, actions, and goals
- Issues for selecting a language to represent the logical structure of the problem
  - Expressive enough to describe a wide variety of problems
  - Restrictive enough to allow efficient algorithms to operate over it
- The STRIPS language
  - **S**tanford **R**esearch **I**nstitute **P**roblem **S**olver
  - A basic representation language of classical planner
    - Tidily arranged actions descriptions, restricted language

STRIPS Language

- Representation of states
  - Represent a state as a conjunction of positive literals
  - Any conditions not mentioned in a state are assumed false
  - Literals in PL or in FOL and being ground and function-free

$Poor \wedge Unknown$

$At(Plane_1, Melbourne) \wedge At(Plane_2, Sydney)$

- Representation of goals
  - Represent the goal (a partially specified state) as a conjunction of positive ground literals
  - A state satisfies a goal if it contains all the atoms represented in goal (and possible other)

$Rich \wedge Famous$

$At(Plane_2, Tahiti)$

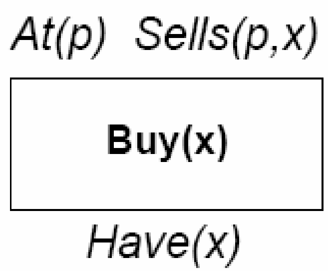
$Rich \wedge Famous \wedge Miserable$

STRIPS Language

- Representation of actions
  - An action is specified in terms of the preconditions and effects
    - Preconditions: state facts must be held before the action
    - Effects: state facts ensued when the action is executed

action schema

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

STRIPS Language

- Action schema consists of three parts
  - Action name and parameter list
    - As the identity of an action
  - Precondition
    - A conjunction of function-free **positive** literals states what must be true in a state before the action can be executed
    - Any variables/terms in the precondition must also appear in the action's parameter list
  - Effect
    - A conjunction of function-free literals states how the state changes when the action is executed
    - Positive literals (in the add list) asserted to be true while negative literals (in the delete list) asserted to be false
    - Variables/terms appear in the effect must also in the action's parameter list

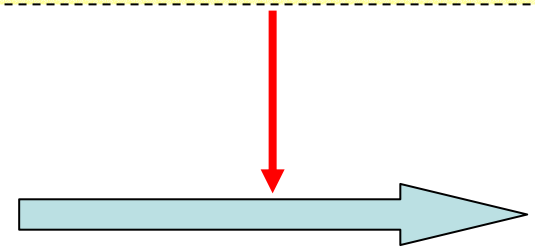
STRIPS Language

- An action is applicable in any state that satisfies the precondition, otherwise the action is has no effect

action schema

Action: *Fly*(*p*, *from*, *to*)
Precondition: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
Effect: $\neg At(p, from) \wedge At(p, to)$

state ***s***



state ***s'***

Positive literals in the effect are added to s' while negative are removed

θ={p/P1, from/JFK, to/SFO}

$At(P_1, JFK) \wedge At(P_2, SFO)$
$\wedge Plane(P_1) \wedge Plane(P_2)$
$\wedge Airport(JFK) \wedge Airport(SFO)$

$At(P_1, SFO) \wedge At(P_2, SFO)$
$\wedge Plane(P_1) \wedge Plane(P_2)$
$\wedge Airport(JFK) \wedge Airport(SFO)$

Example: Air Cargo Transport

$Init(At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$
$\quad \wedge\ Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$
$\quad \wedge\ Airport(JFK) \wedge Airport(SFO))$

$Goal(At(C_1, JFK) \wedge At(C_2, SFO))$

$Action(Load(c, p, a),$
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.

Example: The Spare Tire Problem

$Init(At(Flat, Axle) \wedge At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
  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.

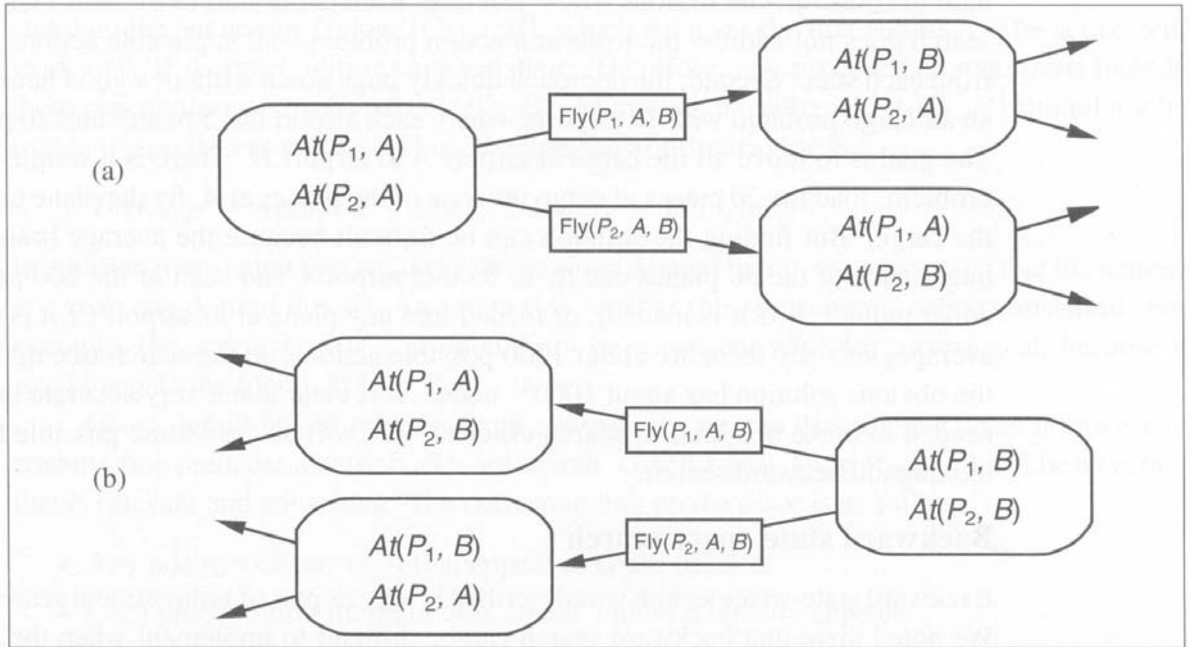
Example: The Blocks World

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

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

Planning with State-Space Search

initial state



goal

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.

Planning with State-Space Search

- Forward state-space search (Progression planning)
  - Start in the problem initial state, consider sequences of actions until find a sequence that reach a goal state
    - Need to face the irrelevant action problem
  - Formulation of planning as state-space search
    - Initial state
      - A set of positive ground literals (literals not appearing are false)
    - Actions
      - Applicable to a state that satisfies the precondition
      - Add positive effect literals to the state presentation and remove the negative ones from it
    - Goal test
      - Check if the state satisfies the goal
    - Step cost
      - Set to unit cost (1) for each action (can be different !)

Planning with State-Space Search

- Backward state-space search (Regression planning)
  - Search backwards from the goal to the initial state
  - Search are restricted to only take the relevant actions
    - A much lower branch factor than forward search

Goal G: $At(C_1, B) \wedge At(C_2, B) \wedge \ldots \wedge At(C_{20}, B)$

Action A: $Unload(C_1, p)$

- Any positive effects of *A* that appear in *G* are deleted
- Each precondition literal of A is added unless it already appears

predecessor : state $In(C_1, p) \wedge At(p, B) \wedge At(C_2, B) \wedge \ldots \wedge At(C_{20}, B)$

must satisfy the preconditions of the action

$\theta=\{p/P1\}$

  - Terminated when a predecessor description is satisfied by the initial state

Heuristics for State-Space Search

- Relaxed-problem heuristic
  - The optimal solution cost for the relaxed problem gives an admissible heuristic for the original problem
  - E.g., remove the all preconditions from the actions (every action will always be applicable)

- Subgoal-independence heuristic
  - The cost of solving a conjunction of subgoals can be approximated by the sum of the costs of solving each subgoal independently
    - Divide-and-conquer $At(C_1, B) \wedge At(C_2, B) \wedge \ldots \wedge At(C_{20}, B)$
  - Could be either optimistic or pessimistic
    - Optimistic: ignore the negative interactions between subplans
    - Pessimistic: ignore the redundant actions between subplans

Heuristics for State-Space Search

$Goal(A \wedge B \wedge C)$
$Action(X, \text{Effect:}A \wedge P)$
$Action(Y, \text{Effect:}B \wedge C \wedge Q)$
$Action(Z, \text{Effect:}B \wedge P \wedge Q)$

– What is the heuristic value ? 2 or 3

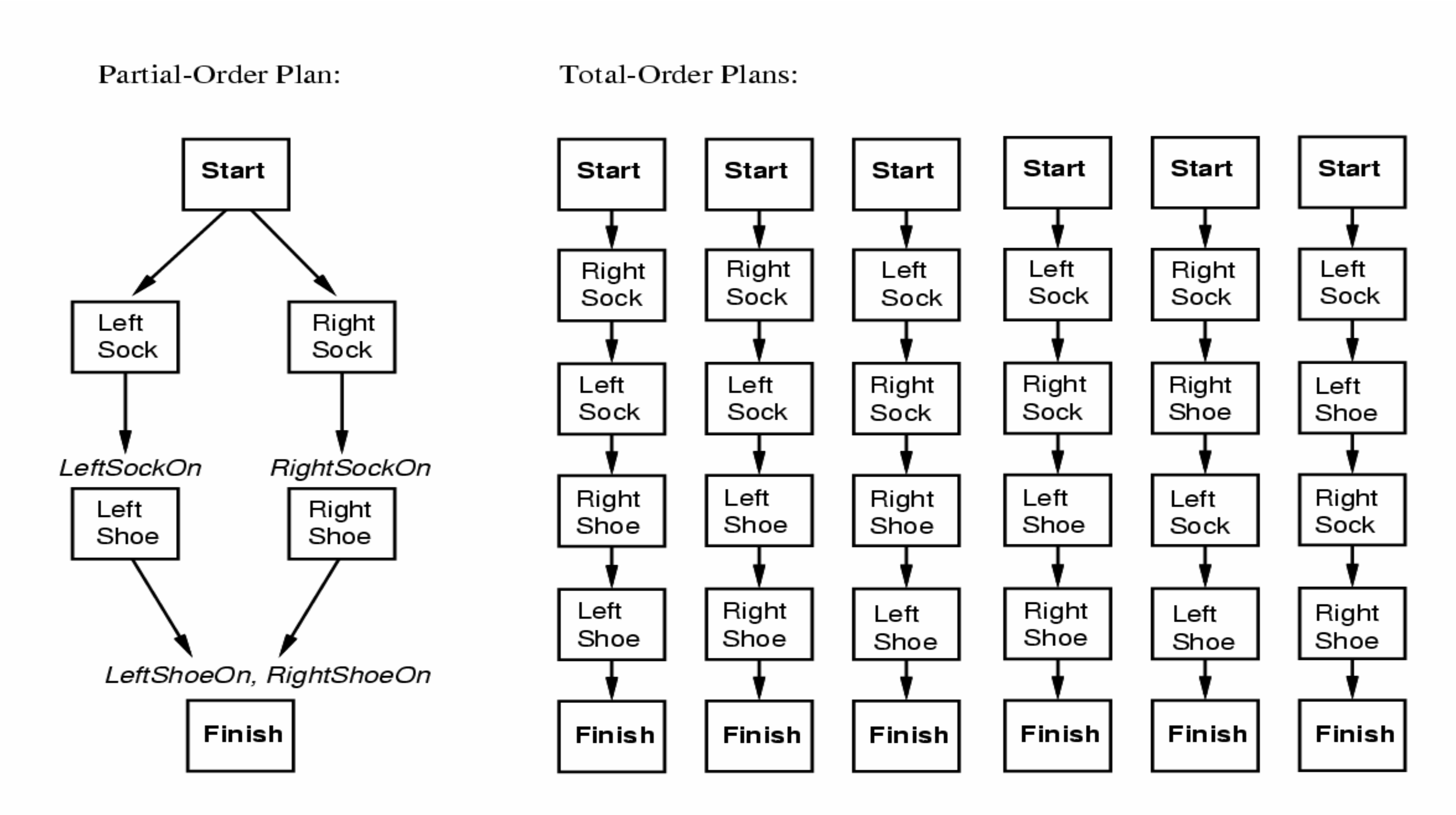
Partial-Order Planning (POP)

- Partial-order planner
  - An planning algorithm that can place two actions in a plan without specifying which comes first
  - Take advantage of problem decomposition
    - Work on subgoals independently

- An example problem

Goal(*RightShoeOn* $\wedge$ *LeftShoeOn*)
Init()
Action(*RightShoe,* PRECOND*: RightSockOn,* EFFECT:*RightShoeOn*)
Action(*RightSock,* EFFECT:*RightSockOn*)
Action(*LeftShoe,* PRECOND*: LeftSockOn,* EFFECT:*LeftShoeOn*)
Action(*LeftSock,* EFFECT:*LeftSockOn*)

Partial-Order Planning



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

- Every step in the plan is an action

Partial-Order Planning

- Partially ordered collection of steps with
  - Start step has the initial state description (literals) as its effect (has no preconditions)
  - Final step has the goal description (literals) as its precondition (has no effects)
  - Causal links from outcome of one step to precondition of another

$$A \xrightarrow{p} B \qquad (A \text{ achieves } p \text{ for } B) \qquad RightSock \xrightarrow{RightSockOn} RightShoe$$

  - Temporal ordering (ordering constraints) between pairs of steps

$$A \prec B \qquad (A \text{ before } B)$$

- Open precondition
  - Precondition of a step not yet causally linked
- A plan is complete iff every precondition is achieved
- A precondition is achieved iff it is the effect of an earlier step and no possibly intervening step undoes it

Partial-Order Planning

$$\text{Actions}: \{RightSock, RightShoe, LeftSock, LeftShoe, Start, Finish\}$$

$$\text{Orderings}: \{RightSock \prec RightShoe, LeftSock \prec LeftShoe\}$$

$$\text{Links}: \{RightSock \xrightarrow{RightSockOn} RightShoe, LeftSock \xrightarrow{LeftSockOn} LeftShoe,$$
$$RightShoe \xrightarrow{RightShoeOn} Finish, LeftShoeShoe \xrightarrow{LeftShoeOn} Finish\}$$

$$\text{Open Preconditions}: \{\ \}$$

- A consistent plan is a plan in which there are no cycles in the ordering constraints and no conflicts with the causal links
  - A consistent plan with no open preconditions is a solution

Partial-Order Planning

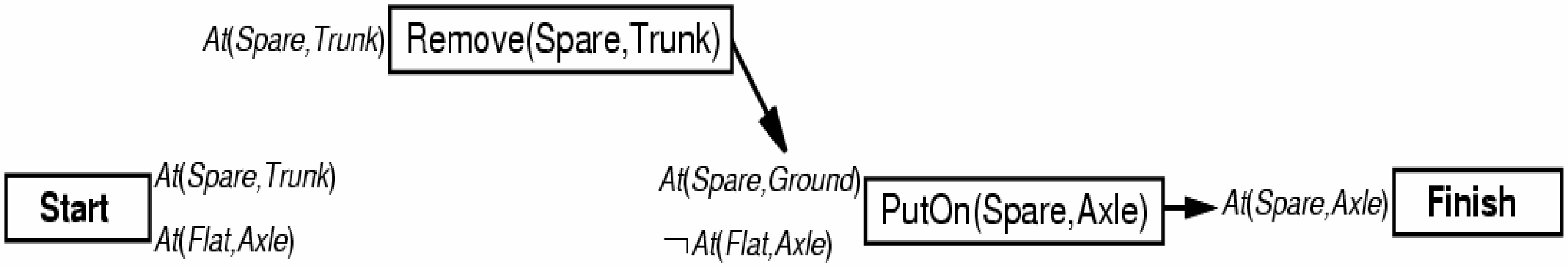
- Formulation of POP search using PL
  - The initial plan contain *Start* and *Finish*, the ordering constraint $Start \prec Finish$, and no causal links and has all the preconditions in *Finish* as open preconditions
  - The successor function arbitrarily picks one precondition *p* on an action *B* and generates a successor plan for every possible consistent way of choosing an action *A* that achieves *p*
    - Need of consistency check
  - Goal test used to check if there are no open preconditions

POP: Flat-Tire Example

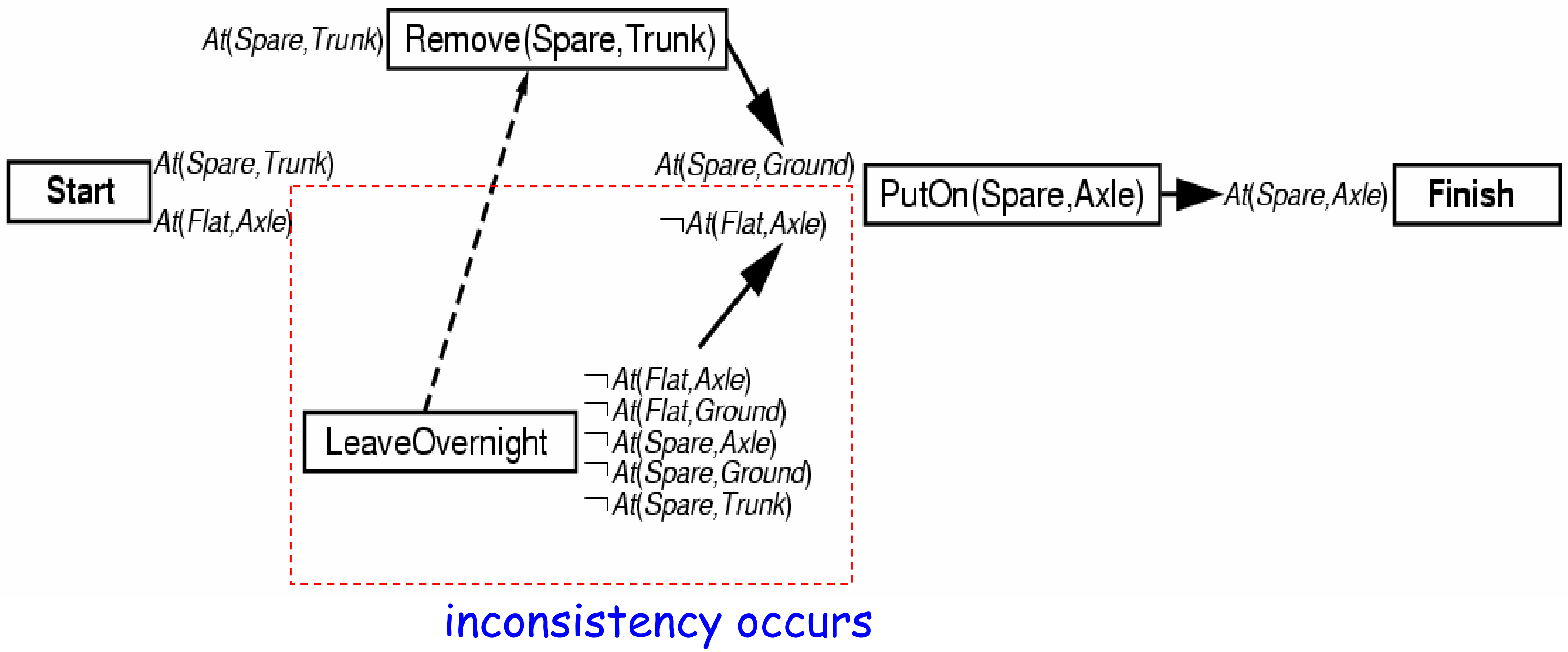
$Init(At(Flat, Axle) \land At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
  PRECOND: $At(Spare, Trunk)$
  EFFECT: $\neg At(Spare, Trunk) \land At(Spare, Ground))$
$Action(Remove(Flat, Axle),$
  PRECOND: $At(Flat, Axle)$
  EFFECT: $\neg At(Flat, Axle) \land At(Flat, Ground))$
$Action(PutOn(Spare, Axle),$
  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 11.7 The simple flat tire problem description.

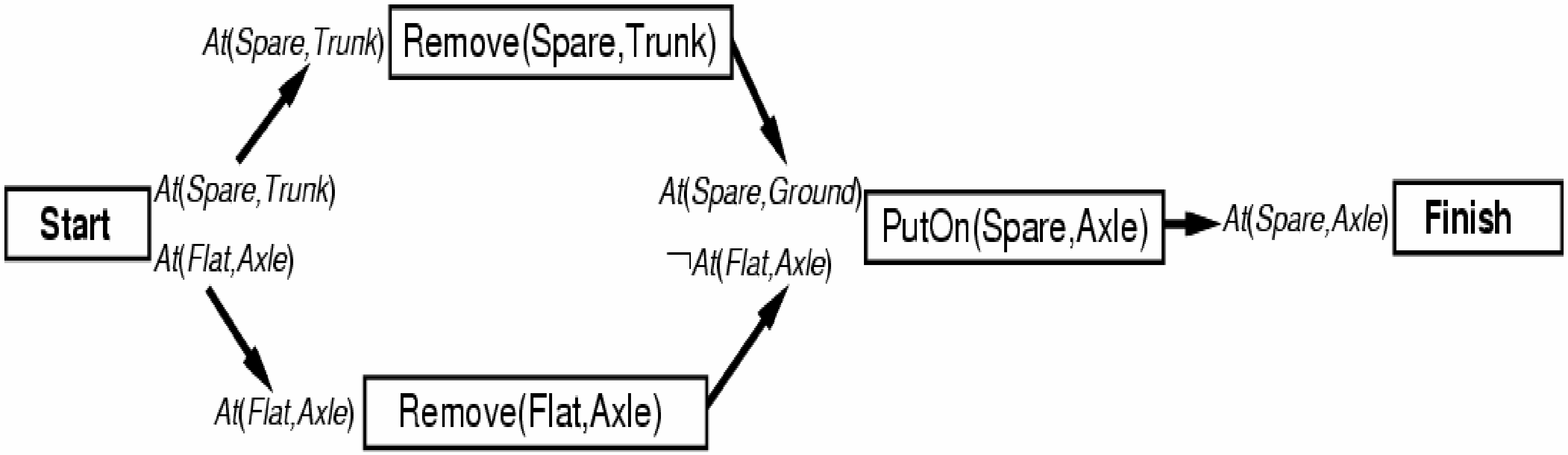
POP: Flat-Tire Example



POP: Flat-Tire Example



POP: Flat-Tire Example



POP Algorithm

function POP(*initial*, *goal*, *operators*) **returns** *plan*

  plan ← MAKE-MINIMAL-PLAN(*initial*, *goal*)
  loop do
    if SOLUTION?(*plan*) **then return** *plan*
    S_{need}, c ← SELECT-SUBGOAL(*plan*)
    CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, c)
    RESOLVE-THREATS(*plan*)
  end

---

function SELECT-SUBGOAL(*plan*) **returns** S_{need}, c

  pick a plan step S_{need} from STEPS(*plan*)
    with a precondition c that has not been achieved
  return S_{need}, c

POP Algorithm

procedure CHOOSE-OPERATOR(*plan*, *operators*, S_{need}, *c*)

  choose a step S_{add} from *operators* or STEPS(*plan*) that has *c* as an effect
  if there is no such step **then fail**
  add the causal link $S_{add} \xrightarrow{c} S_{need}$ to LINKS(*plan*)
  add the ordering constraint $S_{add} \prec S_{need}$ to ORDERINGS(*plan*)
  if S_{add} is a newly added step from *operators* **then**
    add S_{add} to STEPS(*plan*)
    add $Start \prec S_{add} \prec Finish$ to ORDERINGS(*plan*)

---

procedure RESOLVE-THREATS(*plan*)

  for each S_{threat} that threatens a link $S_i \xrightarrow{c} S_j$ in LINKS(*plan*) **do**
    choose either
      Demotion: Add $S_{threat} \prec S_i$ to ORDERINGS(*plan*)
      Promotion: Add $S_j \prec S_{threat}$ to ORDERINGS(*plan*)
    if not CONSISTENT(*plan*) **then fail**
  end