Planning, State Spaces & Partial-Order Planning

Material from Marie desJardin, Jean-Claude Latombe, Lise Getoor; some material adopted from notes by Andreas Geyer-Schulz and Chuck Dyer

1

Bookkeeping

- Dropping HW5
  - Come see me if this is a problem for you
- HW4: Everyone gets 6 points back
  - Info gain doesn't give the **smallest** tree
  - Finding the actual smallest tree is NP complete
- Projects Phase I due soon
  - If you have questions about your proposal response and haven't talked to me about them yet, please email
- No office hours Monday; email for an appointment if you wish

2

Overview

- What is planning?
- Approaches to planning
  - GPS / STRIPS
  - Situation calculus formalism [revisited]
  - Partial-order planning
  - Hierarchical planning

3

Planning Problem

- What is the planning problem?
- Find a **sequence of actions** that achieves a **goal** when executed from an **initial state**.
- That is, given
  - A set of operators (possible actions)
  - An initial state description
  - A goal (description or conjunction of predicates)
- Compute a sequence of operations: a **plan**.

4

Planning Problem

- What is the planning problem?
- Find a **sequence of actions** that achie[…]xecuted from an **initial state**.
- That is, given
  - A set of operators (possible actions)
  - An initial state description
  - A goal (description or conjunction of predicates)
- Compute a sequence of operations: a[…]

- put on right shoe
- put on left shoe
- put on pants
- put on right sock
- put on left sock
- put on shirt

- pants off
- right shoe off
- right sock off
- right shoe off

(etc)

- pants on
- right shoes on

(etc)

5

Planning vs. Problem Solving

- Planning and problem solving methods can often solve the same sorts of problems
- Planning is more powerful because of the representations and methods used
- States, goals, actions decomposed into sets of sentences
  - Usually in FOL
- Search proceeds through **plan space** rather than **state space**
  - Usually – state space planners exist
- Subgoals can be planned independently, reducing the complexity of the planning problem.

6

Some example domains

- We'll use some simple problems to illustrate planning problems and algorithms
- Putting on your socks and shoes in the morning
  - Actions like put-on-left-sock, put-on-right-shoe
- Planning a shopping trip involving buying several kinds of items
  - Actions like go(X), buy(Y)

7

Typical Assumptions (1)

- **Atomic time:** Each action is indivisible
  - Can't be interrupted halfway through putting on pants
- **No concurrent actions allowed**
  - Can't put on socks at the same time
- **Deterministic actions**
  - The result of actions are completely known – no uncertainty

8

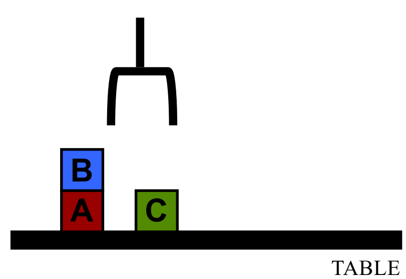
Typical Assumptions

- Agent is the **sole cause of change** in the world
  - Nobody else is putting on your socks
- Agent is **omniscient**:
  - Has complete knowledge of the state of the world
- **Closed world assumption**:
  - Everything known-true about the world is in the state description
  - Anything not known-true is known-false

9

Blocks World

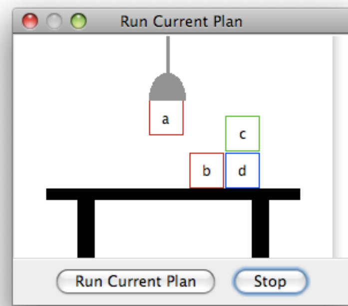
- The **blocks world** consists of a table, set of blocks, and a robot gripper
- Some domain constraints:
  - Only one block on another block
  - Any number of blocks on table
  - Hand can only hold one block
- Typical representation:
  - ontable(a) handempty
  - ontable(c) on(b,a)
  - clear(b) clear(c)



10

Blocks World

- A micro-world
- Some domain constraints:
  - Only one block can be on another block
  - Any number of blocks can be on the table
  - The hand can only hold one block



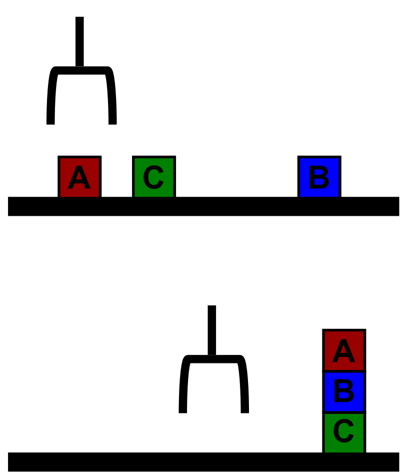
Meant to be a simple model!
Try demo at:
http://aispace.org/planning/

11

Typical BW planning problem

Initial state:
- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal state:
- on(b,c)
- on(a,b)
- ontable(c)

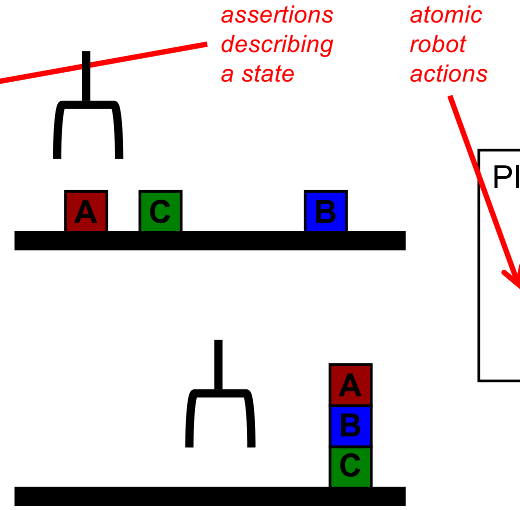


12

Typical BW planning problem

Initial state:
- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal state:
- on(b,c)
- on(a,b)
- ontable(c)



Plan:
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

13

Major Approaches

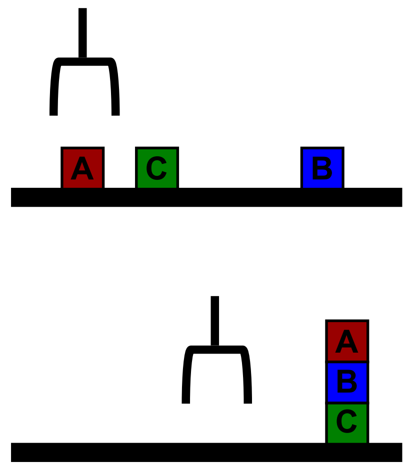
- GPS (General Problem Solver) / STRIPS (Stanford Research Institute Problem Solver)
- **Situation calculus**
- **Partial order planning**
- Hierarchical decomposition (HTN planning)
- Planning with constraints (SATplan, Graphplan)
- **Reactive planning**

14

Another BW planning problem

Initial state:
- clear(a)
- clear(b)
- clear(c)
- ontable(a)
- ontable(b)
- ontable(c)
- handempty

Goal:
- on(a,b)
- on(b,c)
- ontable(c)



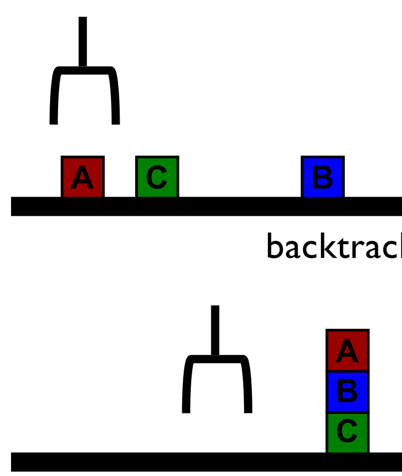
A plan
- pickup(a)
- stack(a,b)
- unstack(a,b)
- putdown(a)
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

15

Yet another BW planning problem

Initial state:
- clear(c)
- ontable(a)
- on(b,a)
- on(c,b)
- handempty

Goal:
- on(a,b)
- on(b,c)
- ontable(c)



Plan:
- unstack(c,b)
- putdown(c)
- unstack(b,a)
- putdown(b)
- putdown(b)
- backtracking {
  - pickup(a)
  - stack(a,b)
  - unstack(a,b)
  - putdown(a)
- }
- pickup(b)
- stack(b,c)
- pickup(a)
- stack(a,b)

16

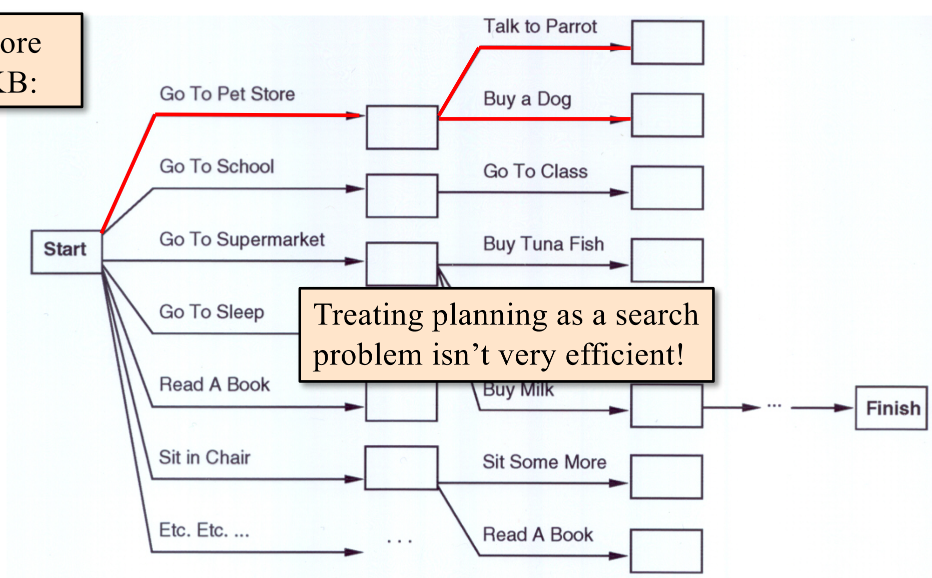
Planning as Search

- Can think of planning as a search problem
  - **Actions:** generate successor states
  - **States:** completely described & only used for successor generation, heuristic fn. evaluation & goal testing
  - **Goals:** represented as a goal test and using a heuristic function
  - **Plan representation:** unbroken sequences of actions forward from initial states or backward from goal state

17

"Get a quart of milk, a bunch of bananas and a variable-speed cordless drill."

Slightly more complex KB:



Treating planning as a search problem isn't very efficient!

18

General Problem Solver

- The **General Problem Solver (GPS)** system
  - An early planner (Newell, Shaw, and Simon)
- Generate actions that *reduce difference* between current state and goal state
- Uses *Means-Ends Analysis*
  - Compare what is given or known with what is desired
  - Select a **reasonable** thing to do next
  - Use a table of differences to identify procedures to reduce differences
- GPS is a **state space planner**
  - Operates on state space problems specified by an initial state, some goal states, and a set of operations

19

Situation Calculus Planning

- Intuition: Represent the planning problem using first-order logic
  - Situation calculus lets us reason about changes in the world
  - Use theorem proving to show ("prove") that a sequence of actions will lead to a desired result, when applied to a world state / situation
- Components:
  - **Initial state:** a logical sentence about (situation) S_0
  - **Goal state:** usually a conjunction of logical sentences
  - **Operators:** descriptions of how the world changes as a result of the agent's actions:
    - Result(a,s) names the situation resulting from executing action a in situation s.
- Action sequences are also useful:
  - Result'(l,s): result of executing list of actions l starting in s

20

Situation Calculus Planning, cont.

- **Initial state**:
  At(Home, S_0) ∧ ¬Have(Milk, S_0) ∧ ¬Have(Bananas, S_0) ∧ ¬Have(Drill, S_0)
- **Goal state**:
  (∃s) At(Home,s) ∧ Have(Milk,s) ∧ Have(Bananas,s) ∧ Have(Drill,s)
- **Operators:**
  ∀(a,s) Have(Milk,Result(a,s)) ⇔
  ((a=Buy(Milk) ∧ At(Grocery,s)) ∨ (Have(Milk, s) ∧ a ≠ Drop(Milk)))
- **Result(a,s)**: situation after executing action a in situation s
  (∀s) Result'([ ],s) = s
  (∀a,p,s) Result'([a|p]s) = Result'(p,Result(a,s))

p=plan

21

Situation Calculus, cont.

- Solution: a **plan** that when applied to the **initial state** gives a situation satisfying the **goal query**:
  At(Home, Result'(p,S_0))
  ∧ Have(Milk, Result'(p,S_0))
  ∧ Have(Bananas, Result'(p,S_0))
  ∧ Have(Drill, Result'(p,S_0))
- Thus we would expect a plan (i.e., variable assignment through unification) such as:
  p = [Go(Grocery), Buy(Milk), Buy(Bananas), Go(HardwareStore), Buy(Drill), Go(Home)]

22

Situation Calculus: Blocks World

- Example situation calculus rule for blocks world:
  - clear(X, Result(A,S)) ↔
    [clear(X, S) ∧
    (¬(A=Stack(Y,X) ∨ A=Pickup(X))
    ∨ (A=Stack(Y,X) ∧ ¬(holding(Y,S))
    ∨ (A=Pickup(X) ∧ ¬(handempty(S) ∧ ontable(X,S) ∧ clear(X,S))))]
    ∨ [A=Stack(X,Y) ∧ holding(X,S) ∧ clear(Y,S)]
    ∨ [A=Unstack(Y,X) ∧ on(Y,X,S) ∧ clear(Y,S) ∧ handempty(S)]
    ∨ [A=Putdown(X) ∧ holding(X,S)]
- English translation: a block is **clear** if
  (a) in the previous state it was clear AND we didn't pick it up or stack something on it successfully, or
  (b) we stacked it on something else successfully, or
  (c) something was on it that we unstacked successfully, or
  (d) we were holding it and we put it down.

Wow.

23

Situation Calculus Planning: Analysis

- Fine in theory, but:
  - Problem solving (search) is exponential in the worst case
  - Resolution theorem proving only finds a proof (plan), not necessarily a good plan
- So what can we do?
  - Restrict the language
    - Blocks world is already pretty small…
  - **Use a special-purpose planner rather than general theorem prover**

24

Basic Representations for Planning

- Classic approach first used in the STRIPS planner circa 1970
- States represented as conjunction of ground literals
  - at(Home) ∧ ¬have(Milk) ∧ ¬have(bananas) ...
- Goals are conjunctions of literals, but may have variables*
  - at(?x) ∧ have(Milk) ∧ have(bananas) ...
- Don't need to fully specify state
  - Un-specified: either don't-care or assumed-false
  - Represent many cases in small storage
  - Often only represent changes in state rather than entire situation
- Unlike theorem prover, not finding whether the goal is true, but whether there is a sequence of actions to attain it

*generally assume ∃

25

Operator/Action Representation

- **Operators** contain three components:
  - **Action description**
  - **Precondition** - conjunction of positive literals
  - **Effect** - conjunction of positive or negative literals which describe how situation changes when operator is applied
- Example:
  - Op[Action: Go(there),
  - Precond: At(here) ∧ Path(here,there),
  - Effect: At(there) ∧ ¬At(here)]

At(here) ,Path(here,there)

Go(there)

At(there) , ¬At(here)

- All variables are universally quantified
- Situation variables are implicit
  - **Preconditions** must be true in the state immediately before operator is applied
  - **Effects** are true immediately after

26

Blocks World Operators

- Classic basic operations for the blocks world:
  - stack(X,Y): put block X on block Y
  - unstack(X,Y): remove block X from block Y
  - pickup(X): pickup block X
  - putdown(X): put block X on the table

(we saw these implicitly in the examples)

- Each will be represented by
  - Preconditions
  - New facts to be added (add-effects)
  - Facts to be removed (delete-effects)
  - A set of (simple) variable constraints (optional!)

27

Blocks World Operators

- So given these operations:
  - stack(X,Y), unstack(X,Y), pickup(X), putdown(X)
- Need:
  - Preconditions, facts to be added (add-effects), facts to be removed (delete-effects), optional variable constraints

Example: stack
preconditions(stack(X,Y), [holding(X), clear(Y)])
deletes(stack(X,Y), [holding(X), clear(Y)]).
adds(stack(X,Y), [handempty, on(X,Y), clear(X)])
constraints(stack(X,Y), [X≠Y, Y≠table, X≠table])

28

Blocks World Operators II

operator(stack(X,Y),
Precond [holding(X), clear(Y)],
Add [handempty, on(X,Y), clear(X)],
Delete [holding(X), clear(Y)],
Constr [X≠Y, Y≠table, X≠table]).

operator(unstack(X,Y),
[on(X,Y), clear(X), handempty],
[holding(X), clear(Y)],
[handempty, clear(X), on(X,Y)],
[X≠Y, Y≠table, X≠table]).

operator(pickup(X),
[ontable(X), clear(X), handempty],
[holding(X)],
[ontable(X), clear(X), handempty],
[X≠table]).

operator(putdown(X),
[holding(X)],
[ontable(X), handempty, clear(X)],
[holding(X)],
[X≠table]).

29

Plan-Space Planning

- Alternative: **search through space of *plans***, not situations
- Start from a **partial plan**; expand and refine until a complete plan that solves the problem is generated
- **Refinement operators** add constraints to the partial plan and modification operators for other changes
- We can still use STRIPS-style operators:
  Op(ACTION: PutOnRightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)
  Op(ACTION: PutOnRightSock, EFFECT: RightSockOn)
  Op(ACTION: PutOnLeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)
  Op(ACTION: PutOnLeftSock, EFFECT: LeftSockOn)

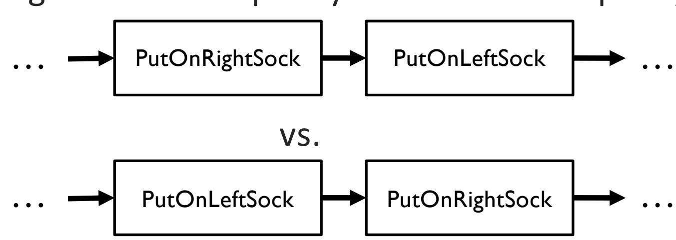
37

Partial-Order Planning

38

Partial-Order Planning

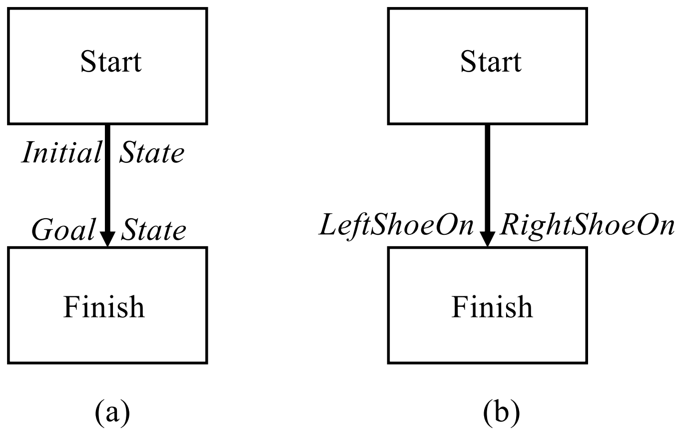
- The big idea: Don't specify the order of steps if you don't have to.



- Doesn't matter, but a regular planner has to consider and specify all the options.

39

A simple graphical notation



40

Partial-Order Planning

- A **linear planner** builds a plan as a **totally ordered sequence** of plan steps
- A **non-linear planner (aka partial-order planner)** builds up a plan as a set of steps with some temporal constraints
  - E.g., S1<S2 (step S1 must come before S2)

PutOnRightSock < PutOnRightShoe

The order here *does* matter, so the planner has to know that.

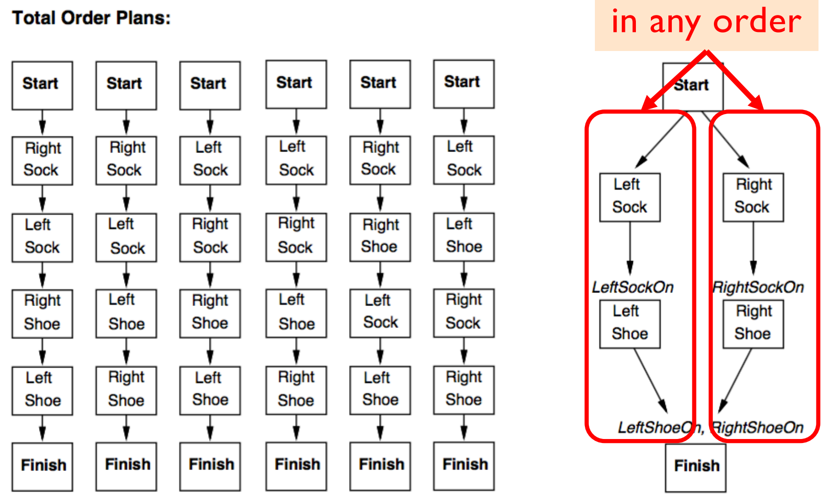
- Partially ordered plan (POP) **refined** by either:
  - adding a new **plan step**, or
  - adding a new **constraint** to the steps already in the plan.
- A POP can be linearized by topological sorting – R&N 223

41

Linear vs. POP: Shoes

- Goal: socks and shoes on

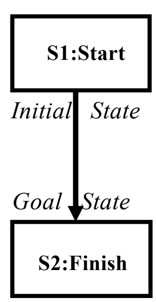
Do these parallel sequences in any order



42

The Initial Plan

- Every plan starts the same way



43

Least Commitment

- Non-linear planners embody the principle of **least commitment**
  - Only choose actions, orderings and variable bindings when absolutely necessary, postponing other decisions
  - Avoid early commitment to decisions that don't really matter
- Linear planners always choose to add a plan step in a particular place in the sequence
- Non-linear planners choose to add a step and possibly some temporal constraints

44

Non-Linear Plan: Completeness

- A non-linear plan consists of
  1. A set of **steps** {S1, S2, S3, S4…}
  2. A set of **causal links** { … (Si,C,Sj) …}
  3. A set of **ordering constraints** { … Si<Sj … }
- A non-linear plan is **complete** iff
  - Every step mentioned in (2) and (3) is in (1)
  - If Sj has prerequisite C, then there exists a causal link in (2) of the form (Si,C,Sj) for some Si
  - If (Si,C,Sj) is in (2) and step Sk is in (1), and Sk threatens (Si,C,Sj) (makes C false), then (3) contains either Sk<Si or Sj<Sk

45

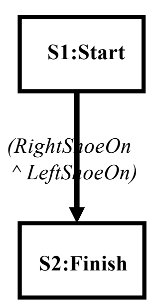
Non-Linear Plan Components

1) A set of **steps** {S_1, S_2, S_3, S_4…}
   - Each step has an **operator description**, **preconditions** and **post-conditions**
   - ACTION: PutOnLeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn
2) A set of **causal links** { … (S_i,C,S_j) …}
   - (One) goal of step S_i is to achieve precondition C of step S_j
   - ⟨PutOnLeftShoe, LeftShoeOn, Finish⟩
     - This says: No action that undoes LeftShoeOn is allowed to happen after PutOnLeftShoe and before Finish. Any action that undoes LeftShoeOn must either be before PutOnLeftShoe or after Finish.
3) A set of **ordering constraints** { … S_i<S_j … }
   - If step S_i must come before step S_j
   - PutOnSock < Finish

46

Trivial Example

- Operators:
  - Op(ACTION: RightShoe, PRECOND: RightSockOn, EFFECT: RightShoeOn)
  - Op(ACTION: RightSock, EFFECT: RightSockOn)
  - Op(ACTION: LeftShoe, PRECOND: LeftSockOn, EFFECT: LeftShoeOn)
  - Op(ACTION: LeftSock, EFFECT: leftSockOn)



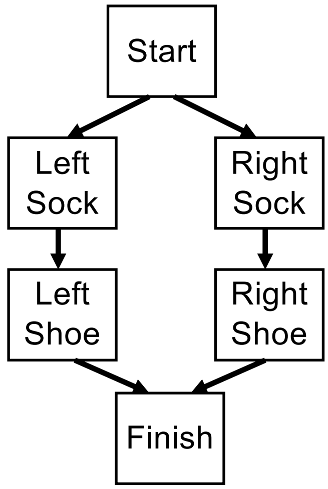
Steps: {S1:[Op(Action:Start)],
S2:[Op(Action:Finish,
Pre: RightShoeOn^LeftShoeOn)]}

Links: {}

Orderings: {S1<S2}

47

Solution



48

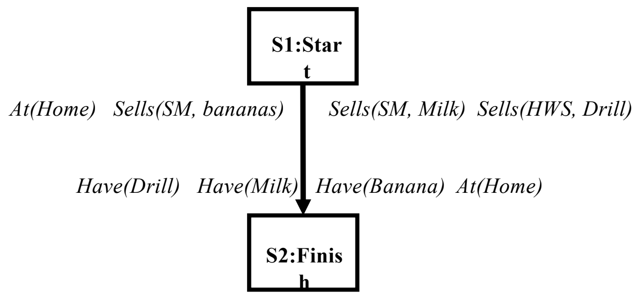
POP Constraints and Search Heuristics

- Only add steps that reach a not-yet-achieved precondition
- Use a least-commitment approach:
  - Don't order steps unless they need to be ordered
- Honor causal links $S_1 \rightarrow S_2$ that **protect** a condition c:
  - Never add an intervening step S_3 that violates c
  - If a parallel action **threatens** c (i.e., has the effect of negating or clobbering c), resolve that threat by adding ordering links:
    - Order S_3 before S_1 (**demotion**)
    - Order S_3 after S_2 (**promotion**)

49

Partial-Order Planning Example

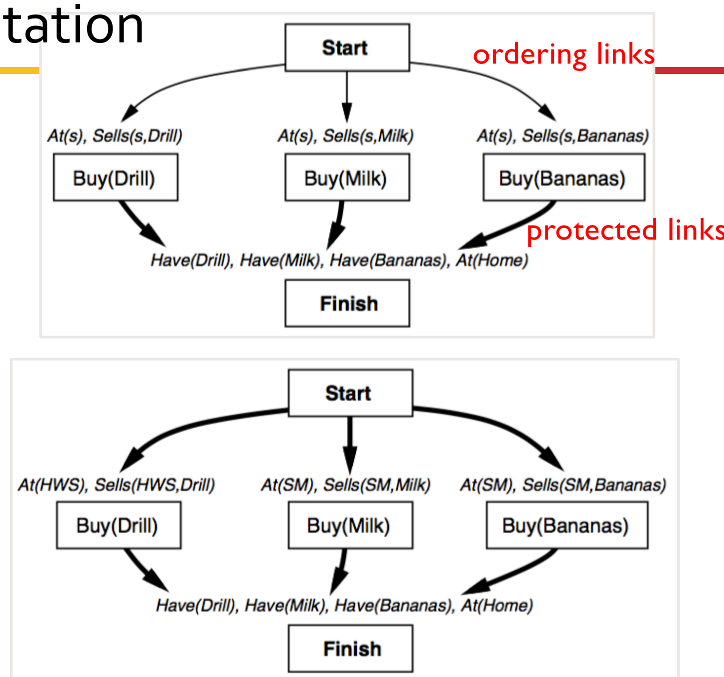
- **Initially**: at home; SM sells bananas; SM sells milk; HWS sells drills
- **Goal**: Be home with milk, bananas, and a drill



51

Graphical representation

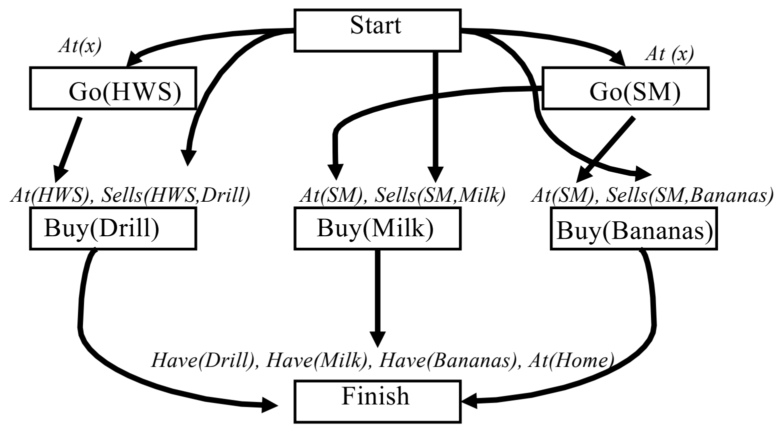
- Add three actions to achieve basic goals
- Use initial state to achieve the "Sells" preconditions
- Bold links are causal (protected), regular are just ordering constraints



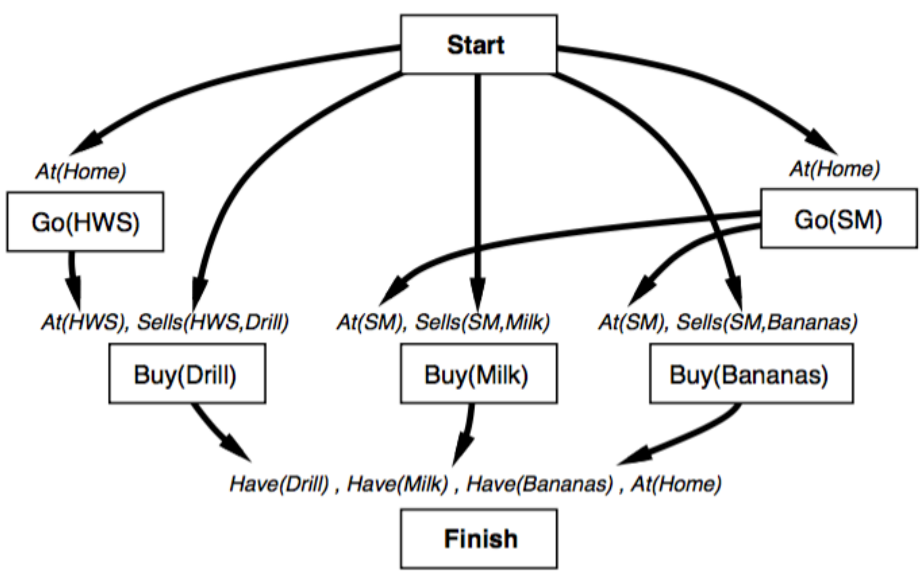
52

Planning

- Hardware store
- Supermarket



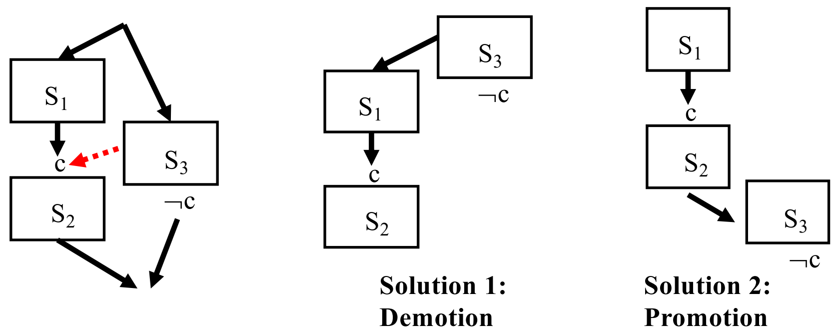
53



54

Resolving Threats

- The S_3 action **threatens** the c precondition of S_2 if S_3 neither precedes nor follows S_2 and S_3 negates c.
  - We don't want to go to the HWS then leave before buying a drill…



55

Partial-Order Planning: Summary

- Idea: plan sequences of actions that don't have temporal constraints with respect to one another – subsequences can happen in any order
- Prevents spending a time searching through spaces that are not meaningfully different
- Plans have steps, causal links, and ordering constraints
- Anything that interferes with the causal links threatens a subsequence and must be demoted or promoted (required to happen before or after the subsequence, but not during)

56

Real-World Planning Domains

- Real-world domains are complex
- Don't satisfy assumptions of STRIPS or partial-order planning methods
- Some of the characteristics we may need to deal with:
  - Modeling and reasoning about resources
  - Representing and reasoning about time
  - Planning at different levels of abstractions
  - Conditional outcomes of actions
  - Uncertain outcomes of actions
  - Exogenous events
  - Incremental plan development
  - Dynamic real-time replanning

Scheduling (modeling and reasoning about resources; representing and reasoning about time; planning at different levels of abstractions)

Planning under uncertainty (conditional outcomes of actions; uncertain outcomes of actions)

HTN planning (incremental plan development; dynamic real-time replanning)

57

Hierarchical Planning

58

Hierarchical Decomposition

- The big idea: **Plan over high-level actions (HLAs), then figure out the steps to accomplish those.**
- Reduces complexity of planning space
  - Consider plan made of HLAs
  - Then make a plan for steps within each
  - Don't consider silly orderings that violate high-level concepts
- Can nest more than one level

59

Hierarchical Decomposition: Example

- If we want to go to Hawaii (and we do)
  - Operators, unordered (because we haven't planned yet): DriveToAirport, TaxiToHotel, PutClothesInSuitcase, BuySunscreen, BoardPlane, BuySwimsuit, FindPassport, PutPassportInCarryon, DisembarkFromPlane, BookHotel …
- High-Level Actions (HLAs): "Get to island" "Prepare for trip"
  - Order HLAs first: PrepareForTrip → GetToIsland
  - THEN order the subgoals within them
  - Don't have to consider "disembark" / "find passport" ordering
- Nest as as needed
  - PrepareForTrip can include ShopForTrip, which includes …

60

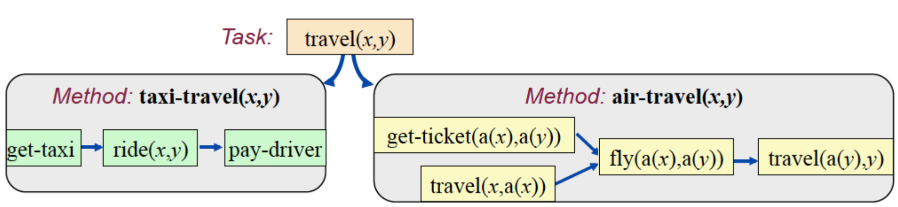
Hierarchical Decomposition

- Hierarchical decomposition, or hierarchical task network (HTN) planning, uses **abstract operators** to **incrementally** decompose a planning problem from a **high-level goal statement** to a **primitive plan network**
- Primitive operators represent actions that are executable, and can appear in the final plan
- Non-primitive operators represent goals (equivalently, abstract actions) that require further decomposition (or operationalization) to be executed
- There is no "right" set of primitive actions: One agent's goals are another agent's actions!

61

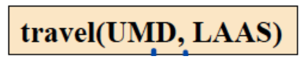
HTN Structure

- Tasks represent recipes for achieving states.
- Primitive tasks are grounded in literals.
- Non-primitive tasks are further decomposed into subtasks subject to constraints.
- Planning is searching through network for a consistent set of tasks to the goal from the initial state.



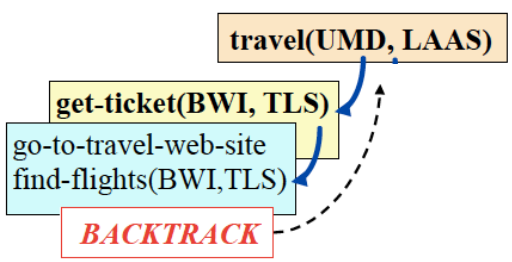
62

Example HTN Search



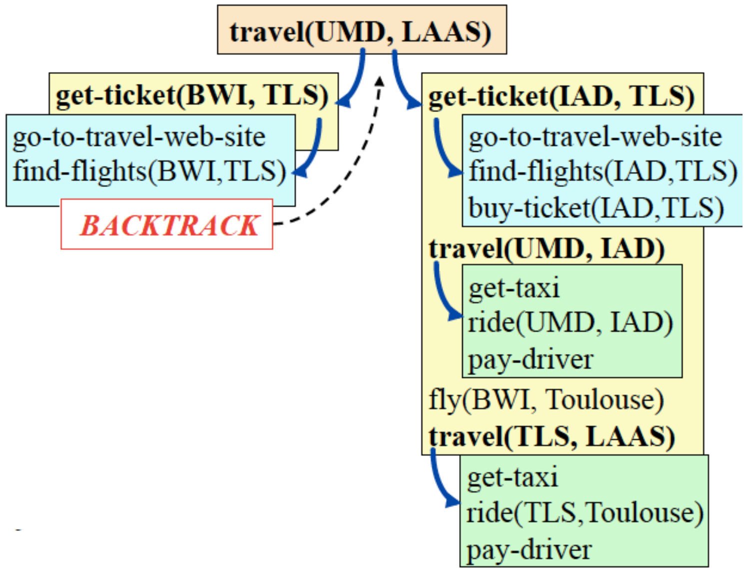
63

Example HTN Search



64

Example HTN Search



65

HTN Operator Representation

- Russell & Norvig explicitly represent causal links
  - Can also be computed dynamically by using a model of preconditions and effects
  - Dynamically computing causal links means that actions from one operator can safely be interleaved with other operators, and subactions can safely be removed or replaced during plan repair
- R&N representation only includes variable bindings
  - Can actually introduce a wide array of variable constraints

68

Truth Criterion

- Determining whether a **formula is true** at a particular point in a partially ordered plan is, in the general case, NP-hard
- Intuition: there are exponentially many ways to **linearize** a partially ordered plan
- In the worst case, if there are N actions unordered with respect to each other, there are N! linearizations
- Ensuring soundness of truth criterion requires checking the formula under all possible linearizations
- Use heuristic methods instead to make planning feasible
- Check later to be sure no constraints have been violated

69

Truth Criterion in HTN Planners

- Heuristic:
  - Prove that there exists one possible ordering of the actions that makes the formula true
  - But don't insert ordering links to enforce that order
- Such a proof is efficient
  - Suppose you have an action A1 with a precondition P
  - Find an action A2 that achieves P (A2 can be initial world state)
  - Make sure there is no action necessarily between A2 and A1 that negates P
- Applying this heuristic for all preconditions in the plan can result in infeasible plans

70

Increasing Expressivity

- Conditional effects
  - Instead of different operators for different conditions, use a single operator with conditional effects
  - Move(block1, from, to) and MoveToTable(block1, from) collapse into one Move(block1, from, to):
    - Op(ACTION: Move(block1, from, to),
      PRECOND: On (block1, from) ^ Clear (block1) ^ Clear (to)
      EFFECT: On (block1, to) ^ Clear (from) ^ ~On(block1, from) ^ ~Clear(to)
      when to<>Table
    - There's a problem with this operator: can you spot it?
- Negated and disjunctive goals
- Universally quantified preconditions and effects

71

Reasoning About Resources

- What if I only have so much money for bananas and drills?
  - It suddenly matters that I don't introduce, e.g., BuyGrapes
- Introduce numeric variables that can be used as measures
- These variables represent resource quantities, and change over the course of the plan
- Certain actions produce (increase the quantity of) resources
- Other actions consume (decrease the quantity of) resources
- More generally, may want different types of resources
  - Continuous vs. discrete
  - Sharable vs. nonsharable
  - Reusable vs. consumable vs. self-replenishing

72

Other Real-World Planning Issues

- Conditional planning
- Partial observability
- Information gathering actions
- Execution monitoring and replanning
- Continuous planning
- Multi-agent (cooperative or adversarial) planning

73

POP Summary

- Advantages
  - Partial order planning is sound and complete
  - Typically produces optimal solutions (plan length)
  - Least commitment may lead to shorter search times
- Disadvantages
  - Significantly more complex algorithms
  - Hard to determine what is true in a state
  - Larger search space, since concurrent actions are allowed

74

Case-Based Planning: Adapting old plans

- Storing plans in a library and using them in “similar” situations.
- How to index and retrieve existing plans?
- How to adapt an old plan to solve a new problem?
- Key question: will refitting existing plans save us work?

75

Contingent Planning

- Develop plans that have built-in alternatives based on state-query during plan execution.
- Usually have alternative branches only at points where it is expected to be significant.
- Doesn’t guarantee that plan will execute successfully.

76

Uncertainty and contingencies

- Flat-tire example: testing for hole will be part of the plan.
- Information-gathering step has two outcomes! Previously, we assumed deterministic effects.
- Why planning with information gathering (sensing) is hard
- Also need to deal with broken plans (assumptions, adversaries, unmodeled effects)

77

Conditional planning

Check(Tire1)

If Intact(Tire1) then Inflate(Tire1) else CallAAA

- Separate sub-plans for each contingency.
- Universal or Conformant plans: An extreme form of conditional planning that covers all possible execution-time contingencies.
  - Usually mean forcing environment into a state, e.g. two get two chairs the same color, paint both brown.
- But, planning for many unlikely cases is expensive. Run-time re-planning is an alternative.

78

Re-Planning

- Generate initial plan
- Begin execution of the plan and monitor each step.
- Check for inconsistencies between execution results and planning assumptions.
- Replan when inconsistencies are detected.

What are the dangers of replanning?

79

Multi-Agent Planning

- Instead of centralized plan, now we have a coordinated plan based on individual agents committing to actions.
- Agents have to negotiate with other agents to determine their actions.
- Different negotiation environments, e.g., self-interested vs. cooperating.

80

Planning Summary

- Planning representations
  - Situation calculus
  - STRIPS representation: Preconditions and effects
- Planning approaches
  - State-space search (STRIPS, forward chaining, ….)
  - Plan-space search (partial-order planning, HTNs, …)
  - *Constraint-based search (GraphPlan, SATplan, …)*
- Search strategies
  - Forward planning
  - Goal regression
  - Backward planning
  - Least-commitment
  - Nonlinear planning

81