POP

Partial Order, Nonlinear Planning

- Two major ideas: least commitment and plan-space search
- *Least commitment:*
  - Don't commit until you have to
  - Commit to what?

Partial Order, Nonlinear Planning

- *Plan-space search:*
  - Search through space of *partial plans*, not states at domain level
  - "States" in this space ≡ partial plans – may be missing steps, ordering constraints, etc.
  - Not $state_1 \longrightarrow state_2$, but $partialPlan_1 \longrightarrow partialPlan_2$
  - Allows planner to focus on what makes sense during planning – may switch from one goal to another, e.g.
  - Operators: add domain-level operator, impose order on the steps, instantiate a variable (or separate variables)

What Constitutes a Solution?

- *Complete* – every precondition for every operator is achieved and no operator removes the precondition required for some operator
- *Consistent* – no contradictions in ordering or binding constraints
- Order does not need to be complete
  - partial ordering allows operators to be executed in parallel
  - agent may have better information for ordering at execution time
  - can produce a total ordering through linearization

Partial-Order Planner (POP)

1. Begin with initial (dummy) plan
2. While open precondition:
   (a) Choose *open* precondition
   (b) Choose action to achieve precondition (fail if cannot)
   (c) If any causal links are threatened then resolve conflicts
3. If no open preconditions, then success, else fail

Partial-Order Planner (POP)

Overview
POP
- POP Overview
- Initial plan
- Choose open pc
- How to satisfy
- Threats
- Thread resolution
- POP example
- POP example
- Spare Tires
- POP'ing Spare Tires
- POP advantages
- Heuristics

1. Begin with initial (dummy) plan
2. While open precondition:
   (a) Choose *open* precondition
   (b) Choose action to achieve precondition (fail if cannot)
   (c) If any causal links are threatened then resolve conflicts
3. If no open preconditions, then success, else fail

Where are the backtrack points?

POP: Initial Plan

Overview
POP
- POP Overview
- Initial plan
- Choose open pc
- How to satisfy
- Threats
- Thread resolution
- POP example
- POP example
- Spare Tires
- POP'ing Spare Tires
- POP advantages
- Heuristics

- The initial plan – start, finish operators
- Start operator: no preconditions, effects are initial state
- Finish operator: no effects, preconditions are goal state(s)

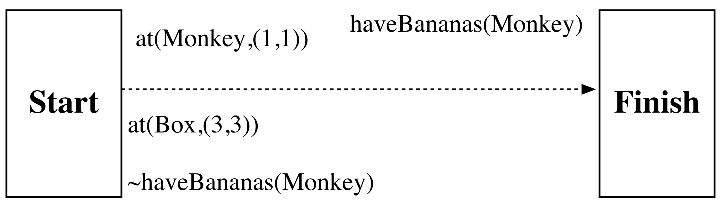
POP: Initial Plan

Overview
POP
- POP Overview
- Initial plan
- Choose open pc
- How to satisfy
- Threats
- Thread resolution
- POP example
- POP example
- Spare Tires
- POP'ing Spare Tires
- POP advantages
- Heuristics

- Example:
  - Initial state:
    $at(Monkey, (1,1)) \wedge at(Box, (3,3)) \wedge$
    $\neg haveBananas(Monkey)$
  - Goal: $haveBananas(Monkey)$

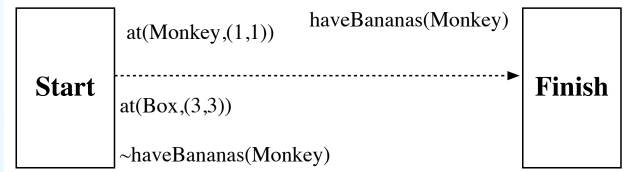
POP: Initial Plan

- Example:
  - Initial state:
    $at(Monkey,(1,1)) \wedge at(Box,(3,3)) \wedge \neg haveBananas(Monkey)$
  - Goal: $haveBananas(Monkey)$



POP: Choosing Open Precondition

- Which actions are needed to achieve preconditions (PCs) is recorded by *causal links*
- Choose a precondition to work on that currently has nothing linked to it
- E.g., for:



choose $haveBananas(Monkey)$

POP: Choose Way to Satisfy Precondition

- Look through effects of steps already in the plan first – including the Start step
- If find something, add link from that action to PC
- If not, then look for a new operator that can achieve the PC, add it to plan, linked to PC
- If still nothing, then fail

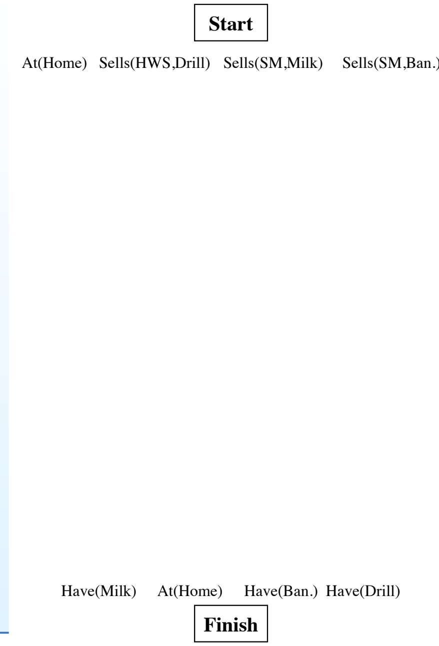
POP: Check for Threats

- Caual links record rationale for plan steps & whether or not PC is open
- E.g., action$_1 \xrightarrow{cond}$ action$_2$ means that action$_1$ is needed to achieve precondition *cond* for action$_2$
- If an action asserts $\neg cond$, then it *threatens* the causal link
- Check for threats when adding a new step:
  - effects may threaten existing link
  - existing actions may threaten new causal link
- Check for threats when using existing action: the new causal link may be threatened by existing actions

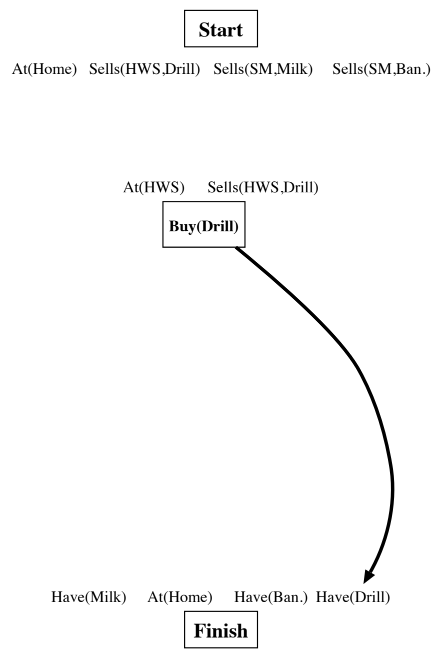
POP: Resolve Threats

- Suppose an action S_3 threatens link $S_1 \xrightarrow{c} S_2$
- Could add order constraints to ensure that S_3 doesn't come between S_1 and S_2
  - *Promotion:* Put S_3 after S_2
  - *Demotion:* Put S_3 before S_1
- If the threat or condition have variables, could also instantiate variables so that there is no conflict (*confrontation*)

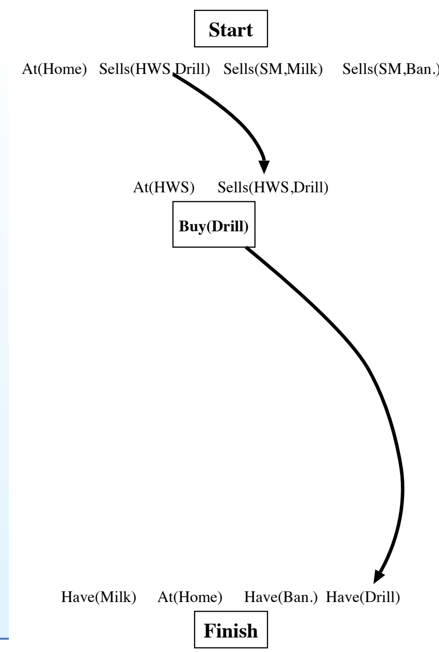
- POP Example: Start state

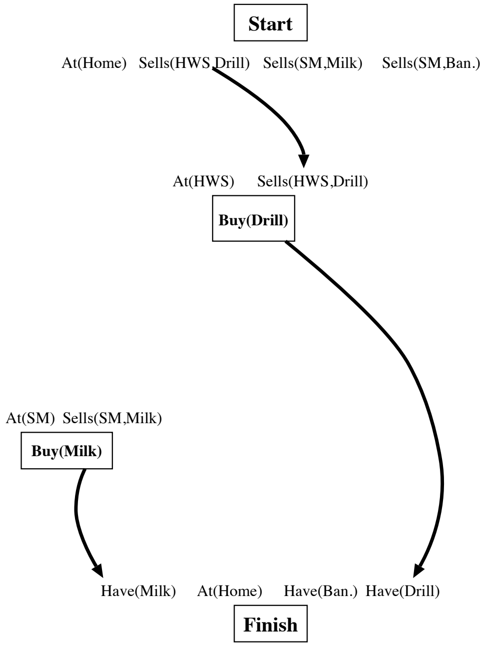


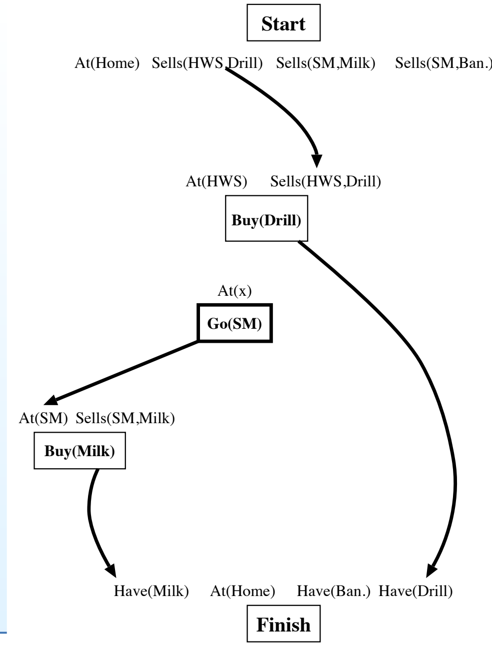
- Add an operator

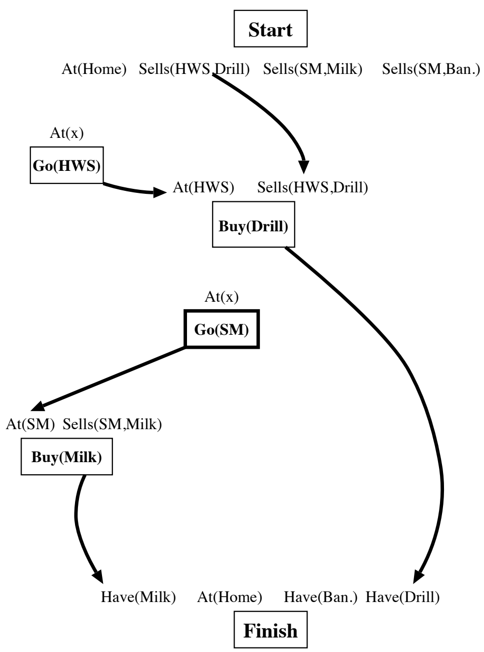


- Use existing action to achieve PC

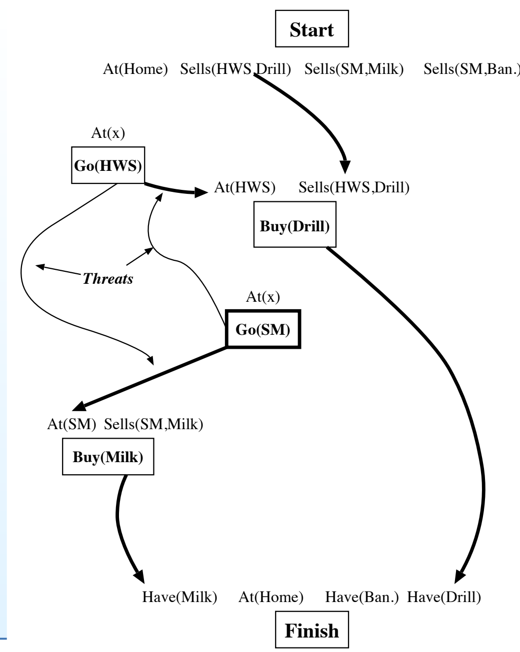



Start
At(Home) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)
At(HWS) Sells(HWS,Drill)
Buy(Drill)
At(SM) Sells(SM,Milk)
Buy(Milk)
Have(Milk) At(Home) Have(Ban.) Have(Drill)
Finish

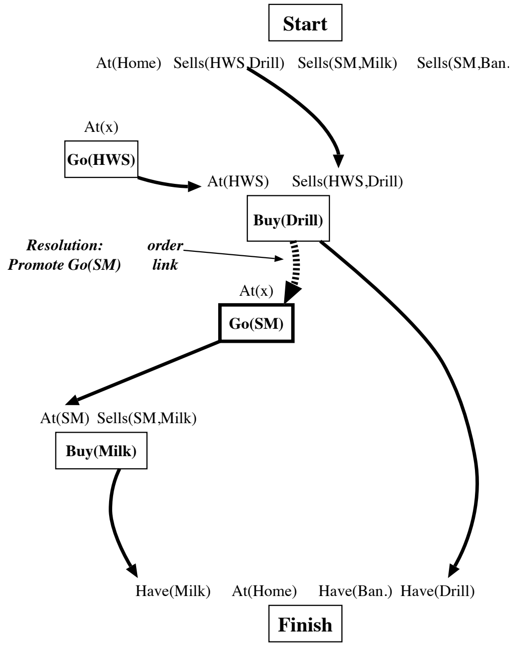

Start
At(Home) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)
At(HWS) Sells(HWS,Drill)
Buy(Drill)
At(x)
Go(SM)
At(SM) Sells(SM,Milk)
Buy(Milk)
Have(Milk) At(Home) Have(Ban.) Have(Drill)
Finish


Start
At(Home) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)
At(x)
Go(HWS)
At(HWS) Sells(HWS,Drill)
Buy(Drill)
At(x)
Go(SM)
At(SM) Sells(SM,Milk)
Buy(Milk)
Have(Milk) At(Home) Have(Ban.) Have(Drill)
Finish

Threats


Start
At(Home) Sells(HWS,Drill) Sells(SM,Milk) Sells(SM,Ban.)
At(x)
Go(HWS)
At(HWS) Sells(HWS,Drill)
Buy(Drill)
Threats
At(x)
Go(SM)
At(SM) Sells(SM,Milk)
Buy(Milk)
Have(Milk) At(Home) Have(Ban.) Have(Drill)
Finish

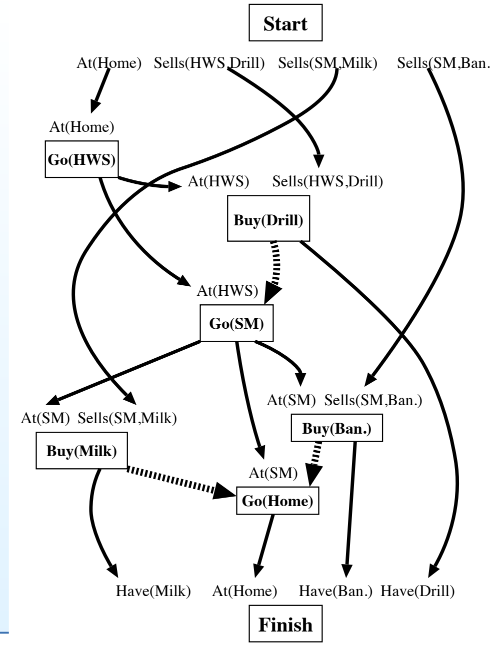
Resolve threats



Final plan

Overview

POP
- POP Overview
- Initial plan
- Choose open pc
- How to satisfy
- Threats
- Thread resolution
- POP example
- POP example
- Spare Tires
- POP'ing Spare Tires
- POP advantages
- Heuristics



Artificial Intelligence

Sussman Anomaly

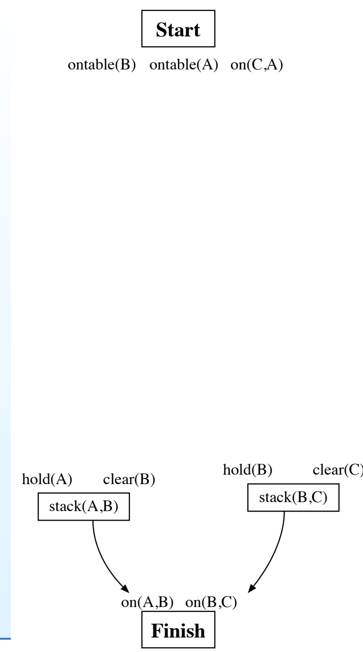
Overview

POP
- POP Overview
- Initial plan
- Choose open pc
- How to satisfy
- Threats
- Thread resolution
- POP example
- POP example
- Spare Tires
- POP'ing Spare Tires
- POP advantages
- Heuristics

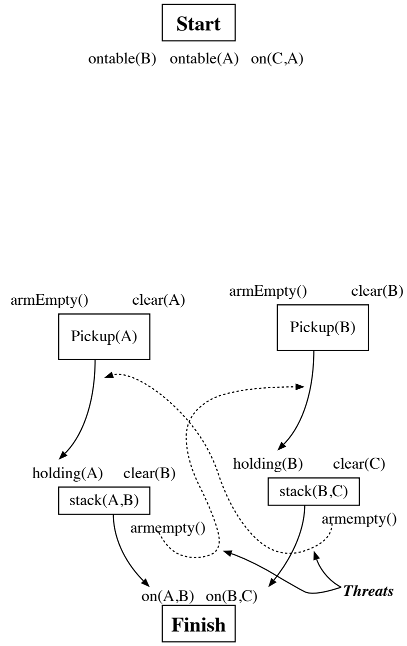
Add operators

Overview

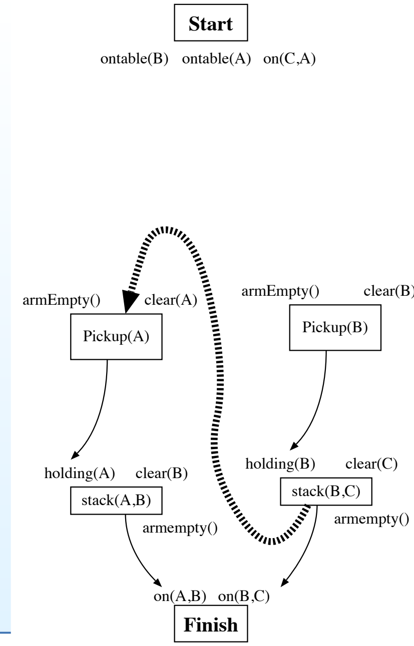
POP
- POP Overview
- Initial plan
- Choose open pc
- How to satisfy
- Threats
- Thread resolution
- POP example
- POP example
- Spare Tires
- POP'ing Spare Tires
- POP advantages
- Heuristics



- Threats



- One way to resolve threats; other would lead to failure & backtracking



Spare Tires

- Spare tire world:

$Init(At(Flat, Axle) \land At(Spare, Trunk))$
$Goal(At(Spare, Axle))$
$Action(Remove(Spare, Trunk),$
  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'ing Spare Tires

- Solution:

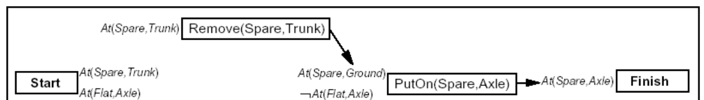


Figure 11.8 The incomplete partial-order plan for the tire problem, after choosing actions for the first two open preconditions. Boxes represent actions, with preconditions on the left and effects on the right. (Effects are omitted, except for that of the *Start* action.) Dark arrows represent causal links protecting the proposition at the head of the arrow.

POP'ing Spare Tires

- Solution:

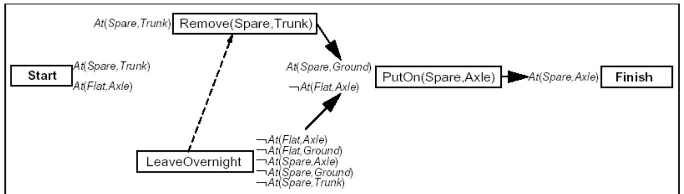


Figure 11.9 The plan after choosing $LeaveOvernight$ as the action for achieving $\neg At(Flat, Axle)$. To avoid a conflict with the causal link from $Remove(Spare, Trunk)$ that protects $At(Spare, Ground)$, $LeaveOvernight$ is constrained to occur before $Remove(Spare, Trunk)$, as shown by the dashed arrow.

POP'ing Spare Tires

Overview

POP
- POP Overview
- Initial plan
- Choose open pc
- How to satisfy
- Threats
- Thread resolution
- POP example
- POP example
- Spare Tires
- POP'ing Spare Tires
- POP advantages
- Heuristics

- Solution:

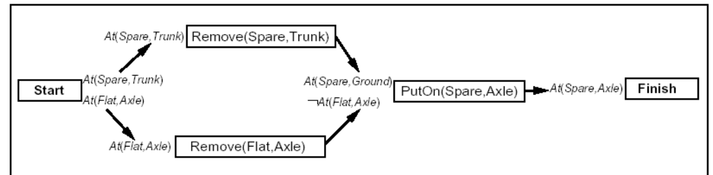


Figure 11.10 The final solution to the tire problem. Note that $Remove(Spare, Trunk)$ and $Remove(Flat, Axle)$ can be done in either order, as long as they are completed before the $PutOn(Spare, Axle)$ action.

Advantages of Partial Order Planning

Overview

POP
- POP Overview
- Initial plan
- Choose open pc
- How to satisfy
- Threats
- Thread resolution
- POP example
- POP example
- Spare Tires
- POP'ing Spare Tires
- POP advantages
- Heuristics

- Reduces backtracking by not commiting until necessary
- Search subplans only where they interact
- Causal links focus on potential problems and show where need to backtrack
- Allows for parallel execution or for ordering steps at execution time

POP Heuristics

Overview

POP
- POP Overview
- Initial plan
- Choose open pc
- How to satisfy
- Threats
- Thread resolution
- POP example
- POP example
- Spare Tires
- POP'ing Spare Tires
- POP advantages
- Heuristics

- Count open preconditions, or open preconditions – preconditions in start state
- Select open precondition that can be satisfied in fewest possible ways (cf. most-constrained variable heuristic from CSP)
- *Planning graphs* good source of heuristics

Plan Graphs

- Levels of graph: each contain states and actions
  - States portion: all possible states that could arise from actions in previous level
  - Actions portion: all actions that could possibly be applied at step i, given the states
- Persistence actions

Plan Graphs (cont'd)

- Mutex links:
  - For actions:
    - inconsistent effects: one action's effects negates effect of another
    - interference: one action negates precondition of another
    - competing needs: two actions require inconsistent states as preconditions
  - For states (literals): if one is negation of other or each possible pair of actions that could achieve the two is mutex
- Leveling off of graph

Cake Eating World

- "Have cake and eat it too" problem:

  $Init(Have(Cake))$
  $Goal(Have(Cake)) \land Eaten(Cake))$
  $Action(Eat(Cake))$
  Precond: $Have(Cake)$
  Effect: $\neg Have(Cake) \land Eaten(Cake)$
  $Action(Bake(Cake))$
  Precond: $\neg Have(Cake)$
  Effect: $Have(Cake)$

Plan Graphs and Cake Eating

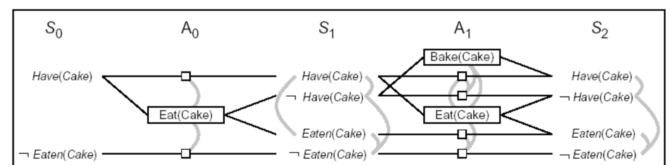


Figure 11.12 The planning graph for the "have cake and eat cake too" problem up to level S_2. Rectangles indicate actions (small squares indicate persistence actions) and straight lines indicate preconditions and effects. Mutex links are shown as curved gray lines.

Plan Graphs and Heuristics

- Can use levels at which precondition appears as estimate of hardness
- Can use *serial planning graph* for better heuristic: insert mutex between all pairs of actions except persistence actions
- Heuristics for computing conjunctive subgoal cost, too

Extracting Plans from Plan Graphs

- In addition to providing heuristics, plan graphs can also be used for planning
- Mutex constraints guide extraction of plan from graph
- Constraint satisfaction techniques can be used to speed up plan extraction
- Creation of plan graph is polynomial-time algorithm – extraction?

Graphplan

Graphplan Algorithm

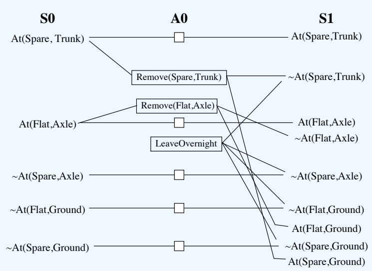
function GRAPHPLAN(*problem*) **returns** solution or failure
  graph ← INITIAL-PLANNING-GRAPH(*problem*)
  goals ← GOALS[*problem*]
  loop do
    if *goals* all non-mutex in last level of *graph* **then do**
      solution ← EXTRACT-SOLUTION(*graph*,*goals*,LENGTH(*graph*))
      if *solution* ≠ *failure*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

Graphplan

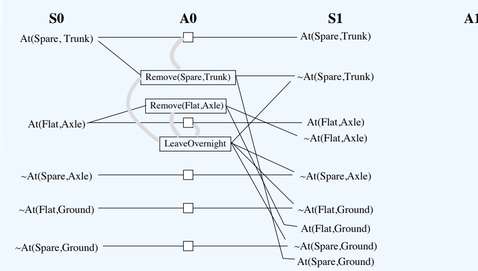
- Termination: when plan is found or planning graph levels off with no solution (approx.)
- Extract-solution:
  - This is the search step
  - For goals at level n, identify consistent subset of actions at level $n-1$ that could produce them
  - Do the same for the preconditions of these actions (at level $n-1$
  - When reach S_0, $\rightarrow$ solution
  - At any level, may need to backtrack
  - Can also approach as a CSP, with actions as variables and values of “in” or “out”
- Very fast planner! Capitalizes on polynomial time to compute graph, plus guidance from plan about what can/cannot happen

Graphplan and Spare Tires

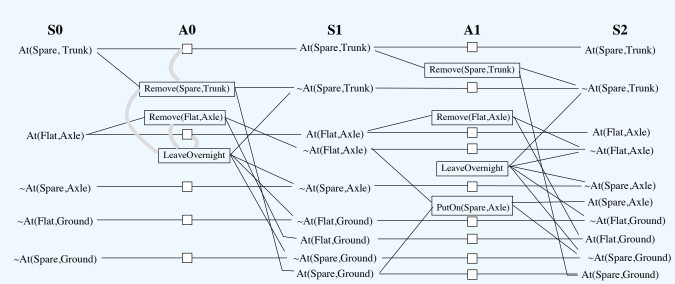
Graphplan and Spare Tires



Graphplan and Spare Tires



Graphplan and Spare Tires

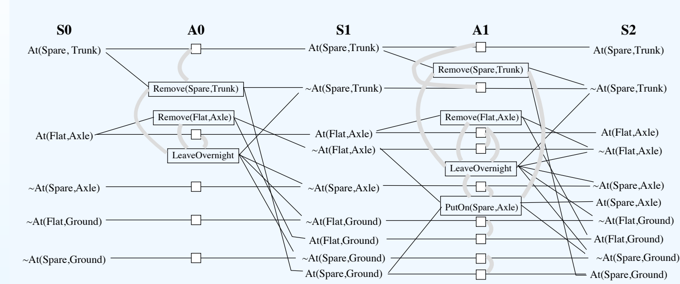


Graphplan and Spare Tires

Plan Graphs

Graphplan
- Algorithm
- Graphplan
- Example
- Problems

Other Planners



Artificial Intelligence

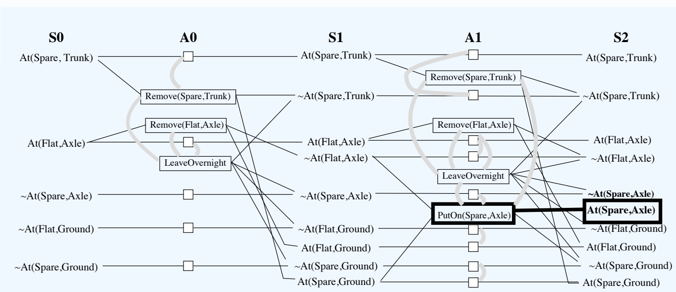
Copyright © 2014 UMaine School of Computing and Information Science – 11 / 15

Graphplan and Spare Tires

Plan Graphs

Graphplan
- Algorithm
- Graphplan
- Example
- Problems

Other Planners



Artificial Intelligence

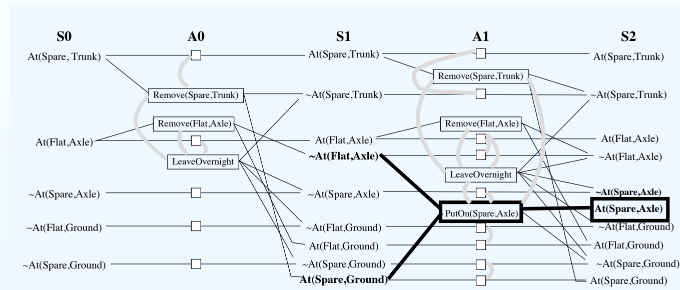
Copyright © 2014 UMaine School of Computing and Information Science – 11 / 15

Graphplan and Spare Tires

Plan Graphs

Graphplan
- Algorithm
- Graphplan
- Example
- Problems

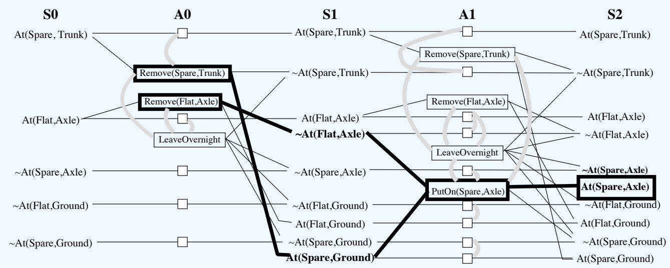
Other Planners



Artificial Intelligence

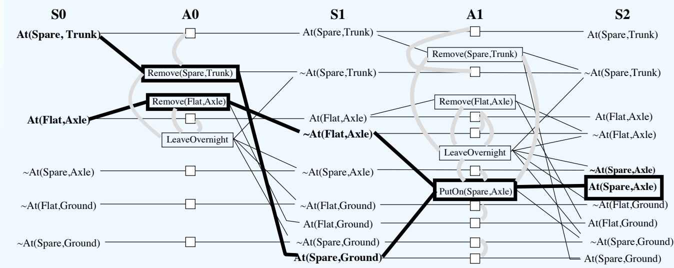
Copyright © 2014 UMaine School of Computing and Information Science – 11 / 15

Graphplan and Spare Tires



Graphplan and Spare Tires

Plan Graphs
Graphplan
• Algorithm
• Graphplan
• Example
• Problems
Other Planners



Artificial Intelligence
Copyright © 2014 UMaine School of Computing and Information Science – 11 / 15

Problems with Graphplan

- Major problem with Graphplan: *propositional planner*
- Potential combinatorial explosion in representation
- There are techniques to reduce this – however, still not scalable (yet) to large, complex domains
- Recently: additions for handling resources, for conditional plans, etc.

Other Planners

Other Forward Planners

- Other graph planners: IPP [Koehler et al.], STAN [Fox, Long], SGP [Weld et al.]
- Satisfiability: SATplan & BlackBox [Kautz, Selman]
- State-space search: UNPOP [McDermott], HSP [Bonet, Geffner], FASTFORWARD (FF) [Hoffmann]

POP'ing Back

- Using CSP, SAT techniques – improve POP
- RePOP [Nguyen and Kambhampati]
- Scales up better than Graphplan

Hierarchical Planning

Problems with Planners Studied So Far

- Focus too much on details
  - E.g., if goal = have(House), plan at level of "swing hammer", ...
  - Leads to very high branching factor, focus on inappropriate details
- Concerned solely with planning – not execution

Hierarchical Decomposition

- Idea: represent steps at different levels of abstraction
  - Some steps: executable actions (e.g., "swing hammer")
  - Other steps: abstract actions (e.g., "put up house frame")
- Advantages:
  - Can focus on outline of plan by dealing with high-level steps...
  - ...lower branching factor
  - Later worry about details after outline okay

Planning with Approximation Hierarchies

- Use the same operators, but check preconditions depending on criticality
  - most critical preconditions checked first
  - plan again, lowering threshold on criticality level each time
- Should find reasons to backtrack quickly because most often caused by most critical preconditions
- Must mark criticality levels for all preconditions on all operators
- Planner using this technique: ABSTRIPS

Hierarchical Planners

- Plan library of plan schemas gives decomposition of step to more detailed representation
- Plan decompositions in library should be well tested
- Have solution when all actions in plans are executable actions
- Need to watch for interactions between steps in different plan schemas
  - critics are daemons that execute to handle specific kinds of interactions
- Planner using this technique: NONLIN [Sacerdoti]

Advantages of Hierarchical Planning

- Plan schemas: same advantages as subroutines
  - can take advantage of cumulative debugging
  - reduced planning effort
- Like top-down programming: Can check whole plan before working with details
- Can focus on most critical steps first
- Saves time and guarantees a solution in certain conditions
  - Downward solution property
  - Upward solution property

Conditional Planning

Conditional/Contingency Planning

- Account for each possibility that may arise
- Operators have conditional steps
  - If C then P else Q
  - P and Q can be lengthy plans
  - Context is the value of conditions needed to get to this step
  - Can have parameterized plans
- Need to have steps to find out value of conditional
- Need to be able to anticipate all possibilities: *universal planning*
- Problems?

Planning and Execution

Adding Execution

- So far: only planning
- Sufficient for some agents
- Other agents need to execute the plans

Execution and Action Monitoring

- One approach: create plan, then monitor its execution
- Two ways: execution or action monitoring
- Execution monitoring:
  - Know which preconditions must be met for each step
  - After current step, see if any are violated (via some possibly complex plan regression)
  - If preconditions not met – have to create situation which meets them (like planning itself)
- Action monitoring:
  - Check actions' effects, not preconditions in general – replan or redo if problem
  - Can also see if there is serendipitous goal satisfaction

What about Unanticipated Events?

- Why do unanticipated events arise?

What about Unanticipated Events?

- Why do unanticipated events arise? CRUD:
  - **C**omplex missions and domains ($\rightsquigarrow$ planning errors, etc.)
  - **R**eal physical systems ($\rightsquigarrow$ imprecision, unpredicted effects)
  - **U**ncertainty
  - **D**ynamic world
- How to handle?
  - Conditional/universal plans
  - Could enumerate events, and specify what to do
  - Could replan or try to repair plan

Combining Planning and Execution

Combining Planning and Execution

- Maybe entire two-phase plan-then-execute is wrong
- Instead, maybe we should put the two together:
  - Can take advantage of delayed/least commitment
  - Can take unanticipated events into account in evolving plan
  - Can avoid creating complex conditional plans

Reactive Planning

- Do not commit to future parts of plan
  - can have schemas for achieving goals, but do not look at future steps to make current decisions
- Does not waste effort on predictive planning when the world is unpredicatable and likely to change between beginning of planning and execution
- Cannot make global optimizations
- Agre & Chapman
- Brooks

Reactive Planning with Goal Schemas

- Place schema on the agenda
- Select a step to execute
- Expand step until reach an executable action
  - at each expansion place steps on agenda to be selected in competition with others – usually use stack-like structure to continue work on same goal
  - choose expansion based on current situation only
- PRS [Georgeff]
- MEDIC, Orca [R. Turner], JUDIS [E. Turner], ACRO [Albert]

Schema-Based Reasoning

Schema-Based Reasoning

- Schema-based reasoning (SBR) [Turner] is an *adaptive reasoning* method
- Adaptive reasoning: agent changes its behavior to fit the evolving problem-solving situation
  - Short-term adaptation
  - Long-term adaptation
  - Adapt in context ⇒ context-mediated behavior (CMB)
- *Schemas* are used to guide reasoning

Schemas

- Schemas are packets of related information used to guide behavior
- Three types:
  - Procedural schemas
  - Contextual schemas
  - Strategic schemas

Procedural Schemas

- Procedural schemas (p-schemas) represent hierarchical plans
- Selection ⇒ partial commitment to a course of action
- Steps can be
  - executable actions
  - other p-schemas
  - sub-goals
- Leave unexpanded until needed ⇒ least commitment

Example


```
(defpschema p-mission (goals)
  :order (sequential analyze-goals preflight-checkout
          launch transit-out work-phase transit-home
          recovery postflight-debrief)
  :steps
    ((analyze-goals (action ^x-analyze-goals)
        (input (?goals => goals))
        (output (location => ?location)
                (equipment => ?equipment)))
     (preflight-checkout ...)
     (launch ...)
     (transit-out
      (action ^p-transit-to)
      (input (location => ?location)))
     (work-phase
      (goals ?goals))
     (transit-home ...) (recovery ...) (postflight-debrief
    ...)
```

Contextual Schemas

- Context-mediated behavior: context should impact all facets of an agent's behavior
- Contextual schemas (c-schemas) represent known contexts
- Process:
  - Retrieve c-schemas that the current situation reminds agent of...
  - Diagnose which one(s) really fit the situation...
  - Merge c-schemas $\Rightarrow$ coherent view of context

Contextual Schemas

- So Far
- Hierarchical Planning
- Conditional Planning
- Planning and Execution
- Combining Planning and Execution
- Schema-Based Reasoning
- Overview
- Schemas
- P-schemas
- C-schemas
- S-schemas
- Process
- Context Assessment: ConMan
- Implementations
- MEDIC
- Orca

- Context provides:
  - Knowledge about the situation
  - Context-specific meaning of symbols, etc.
  - Knowledge about how to handle unanticipated events: how to recognize, how to diagnose, meaning, importance, response
  - Knowledge about goals: which are likely, which are appropriate to pursue
  - Suggestions of actions (p-schemas) to take
- Advantage: automatic context-sensitive reasoning

Example

- So Far
- Hierarchical Planning
- Conditional Planning
- Planning and Execution
- Combining Planning and Execution
- Schema-Based Reasoning
- Overview
- Schemas
- P-schemas
- C-schemas
- S-schemas
- Process
- Context Assessment: ConMan
- Implementations
- MEDIC
- Orca


```
^C-HARBOR is a frame with the following description:
  ISA: (^CONTEXTUAL-SCHEMA)
  SLOTS:
    o ACTORS:
       ((^ACTOR-DESC
           (VARIABLE ?SELF) (BINDING $SELF)
           (DESCRIPTION (^AUV)) (CF 1.0) (PENALTY 1.0)
           (NAME AC1)))
    o OBJECTS:
       ((^OBJECT-DESC
           (VARIABLE ?PLACE) (BINDING $LOCALE)
           (DESCRIPTION (^PLACE)) (CF 1.0) (PENALTY 1.0)
           (NAME OB0))
        (^OBJECT-DESC
           (VARIABLE ?MISSION) (BINDING $MISSION)
           (DESCRIPTION (^MISSION)) (CF 0.5) (NAME OB1))
        (^OBJECT-DESC
           (VARIABLE ?SURFACE) (DESCRIPTION (^SURFACE))
           (NAME OB2))   ...)
```


Example

- So Far
- Hierarchical Planning
- Conditional Planning
- Planning and Execution
- Combining Planning and Execution
- Schema-Based Reasoning
- Overview
- Schemas
- P-schemas
- C-schemas
- S-schemas
- Process
- Context Assessment: ConMan
- Implementations
- MEDIC
- Orca


```
o DESCRIPTION:
   ((^FEATURE-DESC
       (DESCRIPTION (NAME $CONTEXT in harbor))
       (CF 1.0) (NAME FE0))
    (^FEATURE-DESC
       (DESCRIPTION (DEPTH ?WC SHALLOW))
       (CF 0.8) (NAME FE1))
    (^FEATURE-DESC
       (DESCRIPTION
        (AND (TRAFFIC-VOLUME ?SURFACE ?VALUE)
             (>= ?VALUE SOME)))
       (CF 0.7) (NAME FE2))
    ...)
o DEFINITIONS:
   ((^FUZZY-DEFINITION-DESC
     (LINGUISTIC-VARIABLE (SLOT ^PHYSICAL-OBJECT DEPTH))
     (LINGUISTIC-VALUE SHALLOW)
     (MEMBERSHIP-FUNCTION ((0 1) (10 0)))
     (CF 0.8) (COMBINATION-TYPE REPLACE) (NAME FU0))
    ...)
```

Example


```
o EVENTS:
  ((^EVENT-DESC
    (DESCRIPTION (POWER-LEVEL ?SELF LOW))
    (DIAGNOSTIC-INFORMATION NIL)
    (LIKELIHOOD UNLIKELY) (IMPORTANCE CRITICAL)
    (EFFECTS ((^EVENT-DESC (DESCRIPTION
                                (STATUS ?MISSION FAILED))
                               (CF 0.9))
              (^EVENT-DESC (DESCRIPTION
                                 (STATUS ?SELF FAILED))
                               (CF 0.9))))
    (RESPONSE
     (^RESPONSE-DESC (DESCRIPTION (DO (^P-ABORT)))
                         (CF 1.0))) (NAME EV0))
   ...)
o GOALS:
  ((^GOAL-DESC
    (DESCRIPTION (^ACHIEVEMENT-GOAL
                     (STATE (AT ?SELF (?X ?Y 0)))))
    (IMPORTANCE LOW) (NAME GO0))
   ...)
```


Example

- So Far
- Hierarchical Planning
- Conditional Planning
- Planning and Execution
- Combining Planning and Execution
- Schema-Based Reasoning
- Overview
- Schemas
- P-schemas
- C-schemas
- S-schemas
- Process
- Context Assessment: ConMan
- Implementations
- MEDIC
- Orca


```
o STANDING-ORDERS:
  ((^STANDING-ORDER
    (CONDITION T)
    (DESCRIPTION (SET-LLA-PARAMETER DEPTH-ENVELOPE
                                    (5 10)))
    (CF 0.8) (WHEN DURING) (NAME ST0))
   ...)
```


Strategic Schemas

- So Far
- Hierarchical Planning
- Conditional Planning
- Planning and Execution
- Combining Planning and Execution
- Schema-Based Reasoning
- Overview
- Schemas
- P-schemas
- C-schemas
- S-schemas
- Process
- Context Assessment: ConMan
- Implementations
- MEDIC
- Orca

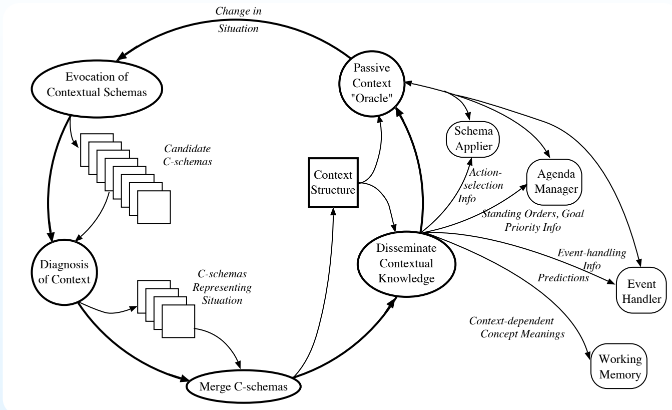
- Strategic schemas (s-schemas) were (and may again be) used to represent an agent's strategies
- E.g., novice versus expert diagnostic reasoning
- Could be just a type of c-schema – unsure at this point what is best

Process

- So Far
- Hierarchical Planning
- Conditional Planning
- Planning and Execution
- Combining Planning and Execution
- Schema-Based Reasoning
- Overview
- Schemas
- P-schemas
- C-schemas
- S-schemas
- Process
- Context Assessment: ConMan
- Implementations
- MEDIC
- Orca

1. Diagnose context (situation/context assessment) – continuous, and in parallel with the rest.
2. Select goal to work on.
3. If no p-schema yet, select one.
4. Expand partially-expanded p-schema to level of finding an executable action
5. Do the action.
6. Go to 2.

Context Assessment: ConMan



Implementations

- Three programs so far
- MEDIC
- Orca
- ACRO

MEDIC

- PhD dissertation work
- Medical diagnosis program: pulmonology
- Modeled after way physicians seem to do their work
- Fundamental contributions:
  - Schema-based reasoning
  - Context-sensitive reasoning (later ⇒ context-mediated behavior)

Orca

- Initially focused on controlling real-world agents: autonomous underwater vehicles
  - Originally: ORCA = Ocean Research Control Architecture...

Orca

- Initially focused on controlling real-world agents: autonomous underwater vehicles
  - Originally: ORCA = Ocean Research Control Architecture...
  - ...but now just Orca...

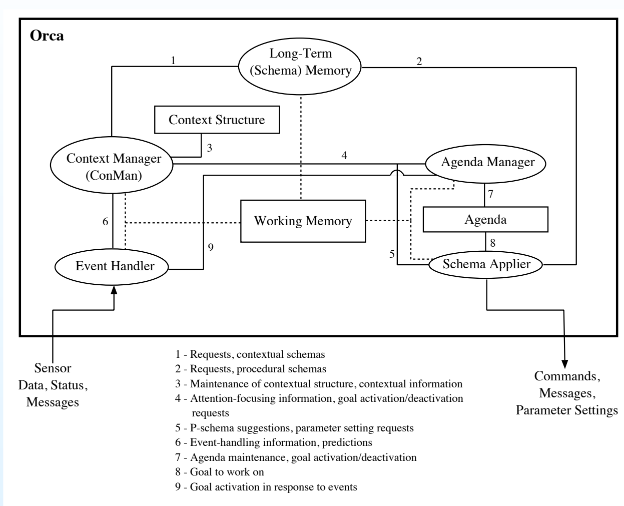
Orca

- So Far
Hierarchical Planning
Conditional Planning
Planning and Execution
Combining Planning and Execution
Schema-Based Reasoning
- Overview
- Schemas
- P-schemas
- C-schemas
- S-schemas
- Process
- Context Assessment: ConMan
- Implementations
- MEDIC
- Orca

- Initially focused on controlling real-world agents: autonomous underwater vehicles
  - Originally: ORCA = Ocean Research Control Architecture...
  - ...but now just Orca...I like orcas...

Orca (Current Version)

- So Far
Hierarchical Planning
Conditional Planning
Planning and Execution
Combining Planning and Execution
Schema-Based Reasoning
- Overview
- Schemas
- P-schemas
- C-schemas
- S-schemas
- Process
- Context Assessment: ConMan
- Implementations
- MEDIC
- Orca



Orca (Next Version)

- So Far
Hierarchical Planning
Conditional Planning
Planning and Execution
Combining Planning and Execution
Schema-Based Reasoning
- Overview
- Schemas
- P-schemas
- C-schemas
- S-schemas
- Process
- Context Assessment: ConMan
- Implementations
- MEDIC
- Orca

- Replace the agenda with an evolving plan template
- Insert new goals and actions into the template
- Focus attention on where in the template should next be expanded, patched, or executed
- Organization of template based on resources, time, etc.: ACRO
- Still guided by context