Fundamentals of Artificial Intelligence
Chapter 11: Planning in the Real World

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it
http://disi.unitn.it/rseba/DIDATTICA/fai_2020/

Teaching assistant: Mauro Dragoni – dragoni@fbk.eu
http://www.maurodragoni.com/teaching/fai/

M.S. Course "Artificial Intelligence Systems", academic year 2020-2021

last update: Monday 30th November, 2020, 13:31

Outline

1. Time, Schedules & Resources
2. Hierarchical Planning
3. Planning & Acting in Non-Deterministic Domains
   - Generalities
   - Sensorless Planning (aka Conformant Planning)
   - Conditional Planning (aka Contingent Planning)

Outline

1. Time, Schedules & Resources

2. Hierarchical Planning

3. Planning & Acting in Non-Determistic Domains
   - Generalities
   - Sensorless Planning (aka Conformant Planning)
   - Conditional Planning (aka Contingent Planning)

Planning with Time, Schedules and Resources

- Planning so far: choice of actions
- Real world: Planning with time/schedules
  - actions occur at certain moments in time
  - actions have a beginning and an end
  - actions have a duration

  $\Longrightarrow$ Scheduling
- Real world: Planning with resources
  - actions may require resources
  - ex: limited number of staff, planes, hoists, ...
- Preconditions and effects can include
  - logical inferences
  - numeric computations
  - interactions with other software packages
- Approach "plan first, schedule later":
  - planning phase: build a (partial) plan, regardless action durations
  - scheduling phase: add temporal info to the plan, s.t. to meet resource and deadline constrains

Planning with Time & Resources: Example

Planning Phase

$Init(Chassis(C1) \land Chassis(C2) \land Engine(E1, C1, 30) \land$
$\quad Engine(E1, C2, 60) \land Wheels(W1, C1, 30) \land Wheels(W2, C2, 15))$
$Goal(Done(C1) \land Done(C2))$
$Action(AddEngine(e, c, d)$
$\quad PRECOND : Engine(e, c, d) \land Chassis(c) \land \neg EngineIn(c)$
$\quad EFFECT : EngineIn(c) \land Duration(d)$
$\quad Consume : LugNuts(20), Use : EngineHoists(1))$
$Action(AddWheels(w, c, d)$
$\quad PRECOND : Wheels(w, c, d) \land Chassis(c)$
$\quad EFFECT : WheelsOn(c) \land Duration(d)$
$\quad Consume : LugNuts(20), Use : WheelStations(1))$
$Action(Inspect(c, 10)$
$\quad PRECOND : EngineIn(c) \land WheelsOn(c) \land Chassis(c)$
$\quad EFFECT : Done(c) \land Duration(10)$
$\quad Use : Inspectors(1))$

Solution (partial plan):

$$\left\{ \begin{array}{l} AddEngine(E1, C1, 30) \prec AddWheels(W1, C1, 30) \prec Inspect(C1, 10); \\ AddEngine(E2, C2, 60) \prec AddWheels(W2, C2, 15) \prec Inspect(C2, 10) \end{array} \right\}$$

Job-Shop Scheduling

- Problem:
  - complete a set of jobs,
  - a job consists of a collection of actions with ordering constraints
  - an action has a duration and is subject to resource constraints
  - resource constraints specify
    - the type of resource (e.g., bolts, wrenches, or pilots),
    - the number of that resource required
    - if the resource is consumable (e.g., bolts) or reusable (e.g. pilot)
    - resources can be produced by actions with negative consumption
- Solution (aka Schedule):
  - specify the start times for each action
  - must satisfy all the temporal ordering constraints and resource constraints
- Cost function
  - may be very complicate (e.g. non-linear constraints)
  - we assume is the total duration of the plan (makespan)

$\Longrightarrow$ Determine a schedule that minimizes the makespan, respecting all temporal and resource constraints

Solving Scheduling Problems

Critical-Path Method

- A path is a ordered sequence of actions from Start to Finish
- The critical path is the path with maximum total duration
  - delaying the start of any action on it slows down the whole plan

  $\Longrightarrow$ determines the duration of the entire plan
  - shortening other paths does not shorten the plan as a whole
- Actions off the critical path have a window of time in which they can be executed: $[ES, LS]$
  - ES: earliest possible start time
  - LS: latest possible start time
  - LS-ES: slack of the action
- LS & ES for all actions can be computed recursively:

$$
\begin{aligned}
ES(Start) &= 0\\
ES(B) &= max_{\{A \mid A \prec B\}}(ES(A) + Duration(A))\\
LS(Finish) &= ES(Finish)\\
LS(A) &= min_{\{B \mid B \succ A\}}(LS(B) - Duration(A))
\end{aligned}
$$

- Complexity: $O(Nb)$, N: #actions, b: max branching factor

Planning with Time & Resources: Example [cont.]

Scheduling Phase

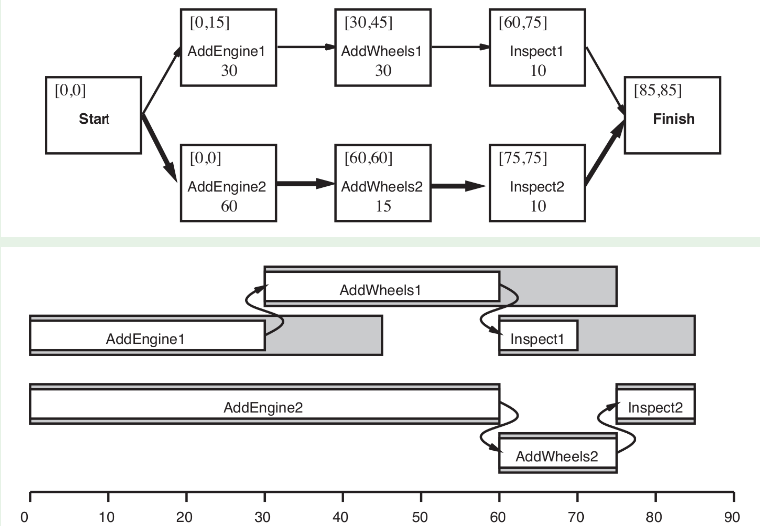
$$Jobs(\{AddEngine1 \prec AddWheels1 \prec Inspect1\},$$
$$\{AddEngine2 \prec AddWheels2 \prec Inspect2\})$$

$$Resources(EngineHoists(1), WheelStations(1), Inspectors(2), LugNuts(500))$$

$$Action(AddEngine1, \text{DURATION}{:}30,$$
$$\text{USE}{:}EngineHoists(1))$$
$$Action(AddEngine2, \text{DURATION}{:}60,$$
$$\text{USE}{:}EngineHoists(1))$$
$$Action(AddWheels1, \text{DURATION}{:}30,$$
$$\text{CONSUME}{:}LugNuts(20), \text{USE}{:}\ WheelStations(1))$$
$$Action(AddWheels2, \text{DURATION}{:}15,$$
$$\text{CONSUME}{:}LugNuts(20), \text{USE}{:}\ WheelStations(1))$$
$$Action(Inspect_i, \text{DURATION}{:}10,$$
$$\text{USE}{:}Inspectors(1))$$

Planning with Time & Resources: Example [cont.]

Scheduling Phase



(© S. Russell & P. Norwig, AIMA)

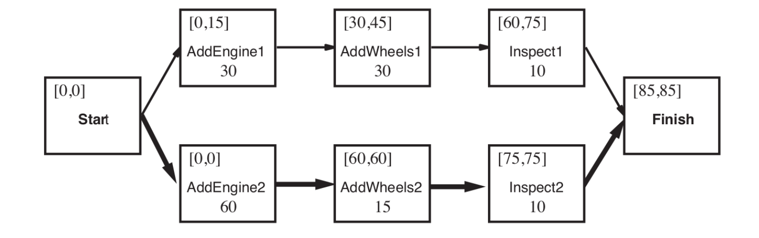
Adding Resources

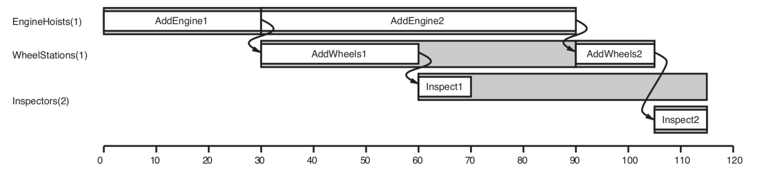
- Critical-path problems (without resources) computationally easy:
  - conjunction of linear inequalities on the start and end times:
    ex: $(ES_2 \geq ES_1 + duration_1) \wedge (ES_3 \geq ES_2 + duration_2) \wedge \ldots$
- Reusable resources: R(k) (ex: Use: EngineHoists(1))
  - k units of resource are required by the action.
  - is a pre-requisite before the action can be performed.
  - resource can not be used for k time units by other.
- Adding resources makes problems much harder
  - "cannot overlap" constraint is disjunction of linear inequalities
    ex: $((ES_2 \geq ES_1 + duration_1) \vee (ES_1 \geq ES_2 + duration_2)) \wedge \ldots$

  $\Longrightarrow$ NP-hard
- Various techniques:
  - branch-and-bound, simulated annealing, tabu search, ...
  - reduction to constraint optimization problems
  - reduction to optimization modulo theories (combined SAT+LP)
- Integrate planning and scheduling

Planning with Time & Resources: Example [cont.]

Scheduling Phase





(© S. Russell & P. Norwig, AIMA)

- left-hand margin lists the three reusable resources
- two possible schedules: which assembly uses the hoist first
- shortest-duration solution, which takes 115 minutes

Exercise

- Consider the previous example
  - find another solution
  - draw the diagram
  - check its length and compare it with that in the previous slide

Outline

1. Time, Schedules & Resources
2. Hierarchical Planning
3. Planning & Acting in Non-Determistic Domains
   - Generalities
   - Sensorless Planning (aka Conformant Planning)
   - Conditional Planning (aka Contingent Planning)

Hierarchical Planning: Generalities

- Real-World planning problems often too complex to handle
- Hierarchical Planners manage the creation of complex plans at different levels of abstraction, by considering the simplest details only after finding a solution for the most difficult ones.
- Hierarchical plan: hierarchy of action sequences (or partial orders) at distinct abstraction levels
  - each action, in turn, can be decomposed further, until we reach the level of actions that can be directly executed
  - designed by hierarchical decomposition (like, e.g., SW design)
- Ex (vacation plan): "Go to San Francisco airport; take Hawaiian Airlines flight 11 to Honolulu; do vacation for two weeks; take Hawaiian Airlines flight 12 back to San Francisco; go home."
  - "Go to San Francisco airport" can be viewed as a planning task

  $\Longrightarrow$ "Drive to the long-term parking lot; park; take the shuttle to the terminal." or (simplier): "take a taxi to San Francisco airport"
  - "Drive to the long-term parking lot": plan a route
- We need a language that enables operators at different levels

High-Level Actions & Refinements

Hierarchical Task Networks (HTN)

- We assume full observability, determinism and the availability of a set of actions (primitive actions, PAs)
- High-level action (HLA):
  - has one or more possible refinements
  - each refinement is a sequence (or p.o.) of actions (PAs or HLAs)
  - may be recursive
- A HLA refinement containing only primitive actions is an implementation of the HLA
- An implementation of a high-level plan is the concatenation/p.o. of implementations of each HLA in the plan.
- A high-level plan achieves the goal from a given state if at least one of its implementations achieves the goal from that state
  - note: "at least one" implementation, not "all" implementations
  - implicitly, we trust our capability to achieve lower-level sub-plans

Q: How do we deal with multiple implementations?

High-Level Actions & Refinements: Examples

$Refinement(Go(Home, SFO),$
$\quad$ STEPS: $[Drive(Home, SFOLongTermParking),$
$\qquad\qquad Shuttle(SFOLongTermParking, SFO)]\ )$
$Refinement(Go(Home, SFO),$
$\quad$ STEPS: $[Taxi(Home, SFO)]\ )$

$Refinement(Navigate([a, b], [x, y]),$
$\quad$ PRECOND: $a = x \ \wedge \ b = y$
$\quad$ STEPS: $[\,]\ )$
$Refinement(Navigate([a, b], [x, y]),$
$\quad$ PRECOND: $Connected([a, b], [a - 1, b])$
$\quad$ STEPS: $[Left, Navigate([a - 1, b], [x, y])]\ )$
$Refinement(Navigate([a, b], [x, y]),$
$\quad$ PRECOND: $Connected([a, b], [a + 1, b])$
$\quad$ STEPS: $[Right, Navigate([a + 1, b], [x, y])]\ )$
...

Searching for Primitive Solutions

Formulation of HTN Planning

- Often formulated with a single "top level" HLA Act s.t.
  - for each a_i, provide one refinement of a_i with steps: $[a_i, Act]$
  - one refinement of Act with empty steps and a goal as precondition

  $\Longrightarrow$ when goal is achieved, do nothing

  hint: "one plan is given by an action, followed by a plan"
- General Algorithm Schema:

  Repeat

  choose an HLA in the current plan

  replace it with one of its refinements

  Until the plan achieves the goas
- Many variants: breadth-first (next slide), depth-first, iterative-deepening, graph-based, ...

Hierarchical Forward-Planning Search

A breadth-first implementation

function HIERARCHICAL-SEARCH(*problem*, *hierarchy*) **returns** a solution, or failure

```
frontier ← a FIFO queue with [Act] as the only element
loop do
    if EMPTY?(frontier) then return failure
    plan ← POP(frontier)   /* chooses the shallowest plan in frontier */
    hla ← the first HLA in plan, or null if none
    prefix,suffix ← the action subsequences before and after hla in plan
    outcome ← RESULT(problem.INITIAL-STATE, prefix)
    if hla is null then   /* so plan is primitive and outcome is its result */
        if outcome satisfies problem.GOAL then return plan
    else for each sequence in REFINEMENTS(hla, outcome, hierarchy) do
        frontier ← INSERT(APPEND(prefix, sequence, suffix), frontier)
```


REFINEMENTS(HLA,OUTCOME,HIERARCHY)
returns a set of action sequences, one for each refinement of the HLA, whose preconditions are satisfied by the specified state: outcome.

Exercise

Consider the refinements of *Go*(*Home*, *SFO*) of last example

- Apply Hierarchical-Search procedure to that example

Remark

The key to HTN planning

The construction of a plan library containing known methods for implementing complex, high-level actions

- One method: learn them via problem-solving experience
- key issue: the ability to generalize the methods that are constructed,
  - eliminating detail that is specific to the problem instance
- Ex: $Drive(Home, ParkingOf(SFO)), Shuttle(ParkingOf(SFO), SFO)])$
  $\Longrightarrow Drive(x, ParkingOf(y)), Shuttle(ParkingOf(y), y)])$

  (See AIMA book, Ch.19 if interested)

Outline

Outline

Generalities [also recall Ch.04]

- Assumptions so far:
  - the environment is deterministic
  - the environment is fully observable
  - the environment is static
  - the agent knows the effects of each action

$\Longrightarrow$ The agent does not need perception:
  - can calculate which state results from any sequence of actions
  - always knows which state it is in
- In the real world, the environment may be uncertain
  - partially observable and/or nondeterministic environment
  - incorrect information (differences between world and model)

$\Longrightarrow$ If one of the above assumptions does not hold, use percepts
  - the agent's future actions will depend on future percepts
  - the future percepts cannot be determined in advance
- Use percepts:
  - perceive the changes in the world
  - act accordingly
  - adapt plan when necessary

Handling Indeterminacy

- Sensorless planning (aka conformant planning):
  find plan that achieves goal in all possible circumstances
  (regardless of initial state and action effects)
  - for environments with no observations
- Conditional planning (aka contingency planning):
  construct conditional plan with different branches for possible contingencies
  - for partially-observable and nondeterministic environments
- Execution monitoring and replanning:
  while constructing plan, judge whether plan requires revision
  - for partially-known or evolving environments

Differences wrt. general Search (Ch.04)

- planners deal with factored representations rather than atomic
- different representation of actions and observation
- different representation of belief states

Open-World vs. Closed-World Assumption

- Classical Planning based on Closed-World Assumption (CWA)
  - states contain only positive fluents
  - we assume that every fluent not mentioned in a state is false
- Sensorless & Partially-observable Planning based on Open-World Assumption (OWA)
  - states contain both positive and negative fluents
  - if a fluent does not appear in the state, its value is unknown
- A belief state is represented by a logical formula (instead of an explicitly-enumerated set of states)

$\Longrightarrow$ The belief state corresponds exactly to the set of possible worlds that satisfy the formula representing it

- The unknown information can be retrieved via sensing actions (aka percept actions) added to the plan

A Case Study

The table & chair painting problem

Given a chair and a table, the goal is to have them of the same color.
In the initial state we have two cans of paint, but the colors of the paint and the furniture are unknown.
Only the table is initially in the agent's field of view

A Case Study [cont.]

The table & chair painting problem [cont.]

- Initial state:

$$Init(Object(Table) \land Object(Chair) \land Can(C1) \land Can(C2) \land InView(Table))$$

- Goal: $Goal(Color(Chair, c) \land Color(Table, c))$
  - recall: in goal, variable c existentially quantified
- Actions:

  $Action(RemoveLid(can),$
  $\quad Precond : Can(can)$
  $\quad Effect : Open(can))$
  $Action(Paint(x, can),$
  $\quad Precond : Object(x) \land Can(can) \land Color(can, c) \land Open(can)$
  $\quad Effect : Color(x, c))$

  c not part of action's variable list (partially observable only)
- Add an action causing objects to come into view (one at a time):

  $Action(LookAt(x),$
  $\quad Precond : InView(y) \land (x \neq y)$
  $\quad Effect : InView(x) \land \neg InView(y))$

A Case Study [cont.]

The table & chair painting problem [cont.]

- Partially-Observable Problems:
  need to reason about percepts obtained during action

$\Longrightarrow$ Augment PDDL with percept schemata for each fluent. Ex:

  - $Percept(Color(x, c),$
    $Precond : Object(x) \wedge InView(x)$
    "if an object is in view, then the agent will perceive its color"

  $\Longrightarrow$ perception will acquire the truth value of $Color(x, c)$, for every x, c

  - $Percept(Color(can, c),$
    $Precond : Can(can) \wedge InView(can) \wedge Open(can)$
    "if an open can is in view, then the agent perceives the color of the paint in the can"

  $\Longrightarrow$ perception will acquire the truth value of $Color(can, c)$, f.e. can, c

- Fully-Observable Problems:

$\Longrightarrow$ Percept schemata with no preconditions for each fluent. Ex:

  - $Percept(Color(x, c))$

- Sensorless Agent: no percept schema

Handling Indeterminacy [cont.]

- Sensorless planning (aka conformant planning):
  find plan that achieves goal in all possible circumstances
  (regardless of initial state and action effects)
  - for environments with no observations
  - ex: "Open any can of paint and apply it to both chair and table"
- Conditional planning (aka contingency planning):
  construct conditional plan with different branches for possible contingencies
  - for partially-observable and nondeterministic environments
  - ex: "Sense color of table and chair;
    if they are the same, then finish, else sense can paint;
    if color(can) =color(furniture) then apply color to other piece;
    else apply color to both"
- Execution monitoring and replanning:
  while constructing plan, judge whether plan requires revision
  - for partially-known or evolving environments
  - ex: Same as conditional, and can fix errors

Outline

1. Time, Schedules & Resources
2. Hierarchical Planning
3. Planning & Acting in Non-Deterministic Domains
   - Generalities
   - Sensorless Planning (aka Conformant Planning)
   - Conditional Planning (aka Contingent Planning)

[Recall from Ch.04]: Search with No Observation

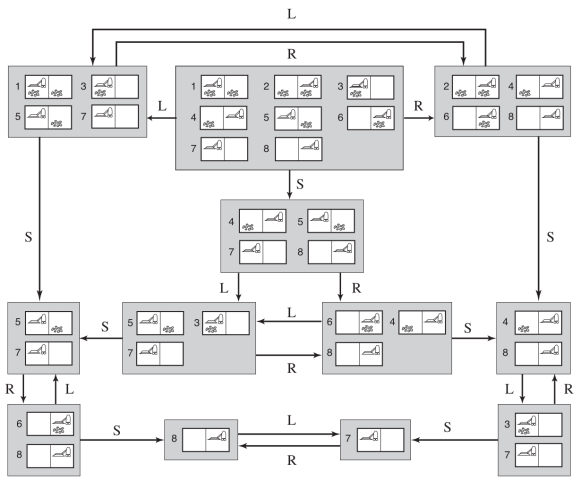
Search with No Observation

- aka Sensorless Search or Conformant Search
- Idea: To solve sensorless problems, the agent searches in the space of belief states rather than in that of physical states
  - fully observable, because the agent knows its own belief space
  - solutions are always sequences of actions (no contingency plan), because percepts are always empty and thus predictable
- Main drawback: 2^N candidate states rather than N

[Recall from Ch.04]: Belief-State Problem Formulation

Example: Sensorless Vacuum Cleaner: Belief State Space

(note: self-loops are omitted)



(© S. Russell & P. Norwig, AIMA)

$\Longrightarrow$ [Left,Suck,Right,Suck] contingent plan

Sensorless Planning

- Main idea [see ch.04]: see a sensorless planning problem as a belief-state planning problem
- Main differences:
  - planners deal with factored representations rather than atomic
  - physical transition model is a collection of action schemata
  - the belief state represented by a logical formula instead of an explicitly-enumerated set of states
- Open-World Assumption $\Longrightarrow$ a belief state corresponds to the set of possible worlds that satisfy the formula representing it
- All belief states (implicitly) include unchanging facts (invariants)
  ex: $Object(Table) \wedge Object(Chair) \wedge Can(C_1) \wedge Can(C_2)$
- Initial belief state includes facts that part of the agent's domain knowledge
  - Ex: "objects and cans have colors"
    $\forall x.\exists c.\ Color(x,c) \Longrightarrow$ (Skolemization)

  $\Longrightarrow$ $b_0 : Color(x, C(x))$

Sensorless Planning [cont.]

- In belief state b, it is possible to apply every action a s.t. $b \models Precond(a)$
  - e.g., $RemoveLid(Can_1)$ applicable in b_0 since $Can(C_1)$ true in b_0
- $Result(b, a)$ is computed:
  - start from b
  - set to false any atom that appears in $Del(a)$ (after unification)
  - set to true any atom that appears in $Add(a)$ (after unification)

  (i.e., conjoint Effects(a) to b)

Property

If the belief state starts as a conjunction of literals, then any update will yield a conjunction of literals

- with n fluents, any belief state can be very-compactly represented by a conjunction of size O(n)

$\Longrightarrow$ much simplifies complexity of belief-state reasoning

Sensorless Planning: Example

- Start from $b_0 : Color(x, C(x))$
- Apply $RemoveLid(Can_1)$ in b_0 and obtain:
  $b_1 : Color(x, C(x)) \wedge Open(Can_1)$
- Apply $Paint(Chair, Can_1)$ in b_1 using $\{x/Can_1, c/C(Can_1)\}$:
  $b_2 : Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1))$
- Apply $Paint(Table, Can_1)$ in b_2 :
  $b_3 : Color(x, C(x)) \wedge Open(Can_1) \wedge Color(Chair, C(Can_1)) \wedge Color(Table, C(Can_1))$
- b_3 Satisfies the goal: $b_3 \models Color(Table, c) \wedge Color(Chair, c)$

$\Longrightarrow$ $[RemoveLid(Can_1), Paint(Chair, Can_1), Paint(Table, Can_1)]$
valid conformant plan

Exercise

- Provide a novel formalization of the above problem with distinct predicates for the color of an object and for the color the paint in a can
    - find step-by-step a plan with the new formalization

Outline

[Recall from Ch.4]:Searching with Nondeterministic Actions

Generalized notion of transition model

- RESULTS(S,A) returns a set of possible outcomes states
  - Ex: RESULTS(1,SUCK)=$\{5,7\}$, RESULTS(5,SUCK)=$\{1,5\}$, ...
- A solution is a contingency plan (aka conditional plan, strategy)
  - contains nested conditions on future percepts (if-then-else, case-switch, ...)
  - Ex: from state 1 we can act the following contingency plan: [SUCK, IF STATE = 5 THEN [RIGHT, SUCK] ELSE [ ]]
- Can cause loops (see later)

[Recall from Ch.4]:Searching with Nondeterministic Actions [cont.]

And-Or Search Trees

- In a deterministic environment, branching on agent's choices

  $\Longrightarrow$ OR nodes, hence OR search trees
  - OR nodes correspond to states
- In a nondeterministic environment, branching also on environment's choice of outcome for each action
  - the agent has to handle all such outcomes

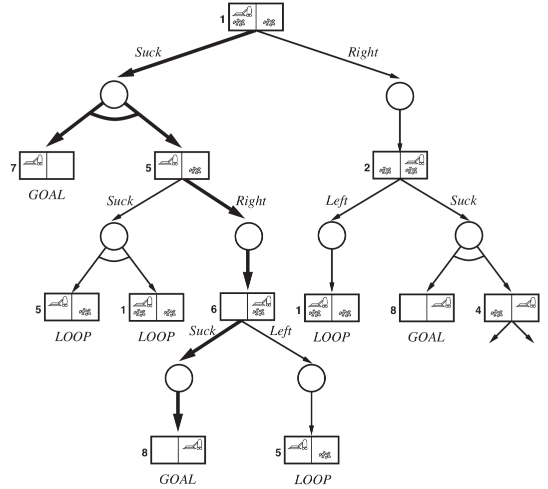
  $\Longrightarrow$ AND nodes, hence AND-OR search trees
  - AND nodes correspond to actions
  - leaf nodes are goal, dead-end or loop OR nodes
- A solution for an AND-OR search problem is a subtree s.t.:
  - has a goal node at every leaf
  - specifies one action at each of its OR nodes
  - includes all outcome branches at each of its AND nodes

OR tree: AND-OR tree with 1 outcome each AND node (determinism)

[Recall from Ch.4]: And-Or Search Trees: Example

(Part of) And-Or Search Tree for Erratic Vacuum Cleaner Example.

Solution for [SUCK, IF STATE = 5 THEN [RIGHT, SUCK] ELSE [ ]]



(© S. Russell & P. Norwig, AIMA)

[Recall from Ch.4]: AND-OR Search

Recursive Depth-First (Tree-based) AND-OR Search

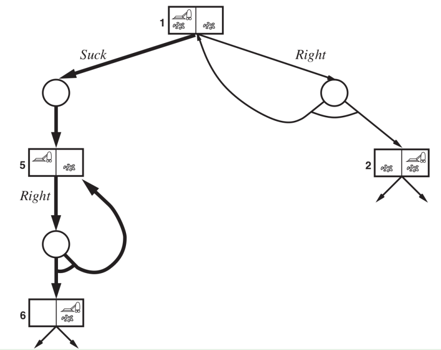
```
function AND-OR-GRAPH-SEARCH(problem) returns a conditional plan, or failure
   OR-SEARCH(problem.INITIAL-STATE, problem, [ ])
---------------------------------------------------------------------------
function OR-SEARCH(state, problem, path) returns a conditional plan, or failure
   if problem.GOAL-TEST(state) then return the empty plan
   if state is on path then return failure
   for each action in problem.ACTIONS(state) do
       plan ← AND-SEARCH(RESULTS(state, action), problem, [state | path])
       if plan ≠ failure then return [action | plan]
   return failure
---------------------------------------------------------------------------
function AND-SEARCH(states, problem, path) returns a conditional plan, or failure
   for each s_i in states do
       plan_i ← OR-SEARCH(s_i, problem, path)
       if plan_i = failure then return failure
   return [if s_1 then plan_1 else if s_2 then plan_2 else ... if s_{n−1} then plan_{n−1} else plan_n]
```


Note: nested if-then-else can be rewritten as case-switch

[Recall from Ch.4]: Cyclic Solution: Example

Example: Slippery Vacuum Cleaner

- Movement actions may fail: e.g., $Results(1, Right) = \{1, 2\}$
- A cyclic solution
- Use labels: [Suck, L1 : Right, if State = 5 then L1 else Suck]
- Use cycles: [Suck, While State = 5 do Right, Suck]



(© S. Russell & P. Norwig, AIMA)

Contingent Planning

- Contingent Planning: generation of plans with conditional branching based on percepts [see Ch.04]
  - appropriate for partial observability, non-determinism, or both
- Main differences:
  - planners deal with factored representations rather than atomic
  - physical transition model is a collection of action schemata
  - the belief state represented by a logical formula instead of an explicitly-enumerated set of states
  - sets of belief states represented as disjunctions of logical formulas representing belief states
- When executing a contingent plan, the agent:
  - maintain its belief state as a logical formula
  - evaluate each branch condition:
    - if the belief state entails the condition formula, then proceed with the "then" branch
    - if the belief state entails the negation of the condition formula, then proceed with the "else" branch
  - Note: The planning algorithm must guarantee that the agent never ends in a belief state where the condition's truth value is unknown

Computing $Result(a, b)$ with Conditional Steps

Three steps (aka prediction-observation-update)

1. Prediction: (same as for sensorless): $\hat{b} = b \setminus Del(a) \cup Add(a)$
2. Observation prediction: determines the set of percepts that could be observed in the predicted belief state
   $P \stackrel{\text{def}}{=} PossiblePercepts(\hat{b}) \stackrel{\text{def}}{=} \{p \mid \hat{b} \models Precond(p)\}$
3. Update: $Result(b, a) = \hat{b} \wedge \bigwedge_{p \in P} b_p$, s.t.:
   - if p has one percept schema, $Percept(p, Precond : c)$, s.t. $\hat{b} \models c$, then $b_p \stackrel{\text{def}}{=} p \wedge c$
   - if p has k percept schemata, $Percept(p, Precond : c_i)$, s.t. $\hat{b} \models c_i$ for each $i = 1..k$, then $b_p \stackrel{\text{def}}{=} \bigvee_{i=1}^{k}(p \wedge c_i)$

$\Longrightarrow$ $Result(b, a)$ CNF formula, not simply conjunction of literals (cubes)

$\Longrightarrow$ much harder to deal with

$\Longrightarrow$ often (over)approximations used to guantantee b_i cube

Contingent Planning: Example

- Possible contingent plan for previous problem described below
  - variables in the plan to be considered existentially quantified
  - ex (2^{nd} row): "if there exists some color c that is the color of the table and the chair, then do nothing" (goal reached)
- "Color(Table,c)", "Color(Chair,c)' and "Color(Can,c)" percepts $\Longrightarrow$ must be matched against percept schemata

$$
\begin{array}{l}
[LookAt(Table), LookAt(Chair), \\
\quad \textbf{if } Color(Table, c) \wedge Color(Chair, c) \textbf{ then } NoOp \\
\qquad \textbf{else } [RemoveLid(Can_1), LookAt(Can_1), RemoveLid(Can_2), LookAt(Can_2), \\
\qquad\qquad \textbf{if } Color(Table, c) \wedge Color(can, c) \textbf{ then } Paint(Chair, can) \\
\qquad\qquad \textbf{else if } Color(Chair, c) \wedge Color(can, c) \textbf{ then } Paint(Table, can) \\
\qquad\qquad \textbf{else } [Paint(Chair, Can_1), Paint(Table, Can_1)]]]
\end{array}
$$

Exercises

- Try to draw an execution the conditiona plan in previous slide against an imaginary phisical state of the wrld of your choice
  - track step by step the belief states, the logical inferences, the actions performed

Is the above plan (from AIMA book) correct?

- If so, explain why it is correct
- If not so, explain why it is not correct, and find a correct one