Planning with Non-Deterministic Uncertainty

(Where failure is not an option)

R&N: Chap. 12, Sect 12.3-5
(+ Chap. 10, Sect 10.7)

Two Cases

- Uncertainty in action only
  [The world is fully observable]
- Uncertainty in both action and sensing
  [The world is partially observable]

Uncertainty in Action Only

Uncertainty Model

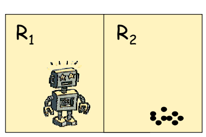
Each action representation is of the form:

Action:
P
$\{E_1, E_2, \ldots, E_r\}$

where each E_i, $i = 1, \ldots, r$ describes one possible effect of executing the action in a state satisfying P

[Using the STRIPS language, E_i consists of a Delete and an Add list]

Example: Devious Vacuum Robot



Right
P = $In(R_1)$
$\{E_1 = [D_1 = In(R_1)$
$A_1 = In(R_2)]$
$E_2 = [D_2 = Clean(R_1)$
$A_2 = \varnothing]\}$

Right may cause the robot to move to room R_2 (E_1), or to dumb dust and stay in R_1 (E_2)

Not intentional, so unpredictable

Left
P = $In(R_2)$
$\{E_1 = [D_1 = In(R_2)$
$A_1 = In(R_1)]\}$

Left always leads the robot to move to R_1

Suck(R_1)
P = In(R_1)
{E_1 = [D_1 = ∅
A_1 = Clean(R_1)]}

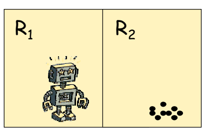
Suck(R_1) always leads the robot to do the right thing

Suck(R_2)
P = In(R_2)
{E_1 = [D_1 = ∅
A_1 = Clean(R_2)]
E_2 = [D_2 = In(R_2)
A_2 = Clean(R_2), In(R_1)]}

But **Suck(R_2)** may also cause the robot to move to R_1

Problem

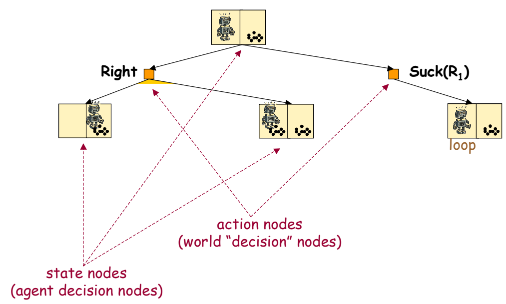
From the initial state:



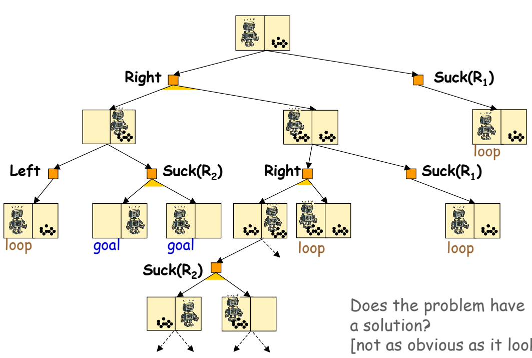
our devious vacuum robot must achieve the goal Clean(R_1) ∧ Clean(R_2)

We want a **guaranteed** plan, i.e., one that works regardless of which action outcomes actually occur

AND/OR Tree

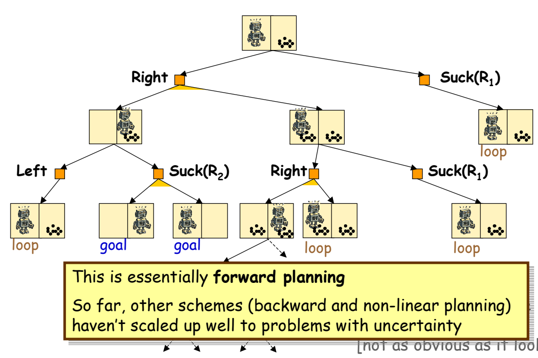


AND/OR Tree



Does the problem have a solution?
[not as obvious as it looks]

AND/OR Tree



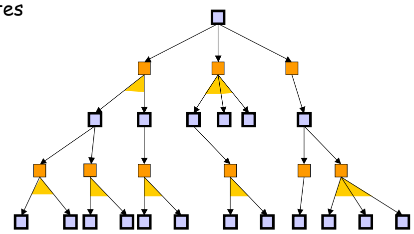
This is essentially **forward planning**

So far, other schemes (backward and non-linear planning) haven't scaled up well to problems with uncertainty

[not as obvious as it looks]

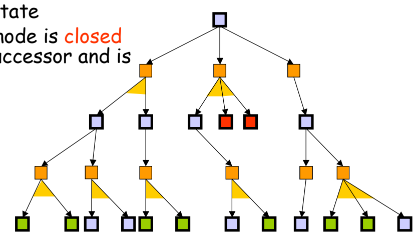
Labeling an AND/OR Tree

- Assume no detection of repeated states



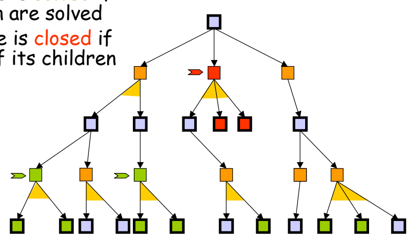
Labeling an AND/OR Tree

- A leaf state node is solved if it's a goal state
- A leaf state node is closed if it has no successor and is not a goal



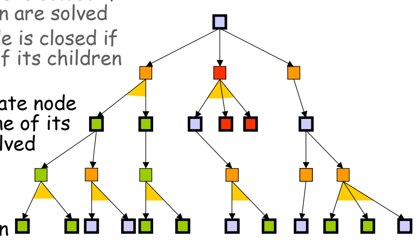
Labeling an AND/OR Tree

- An action node is solved if all its children are solved
- An action node is closed if at least one of its children is closed



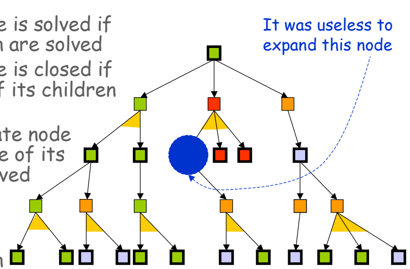
Labeling an AND/OR Tree

- An action node is solved if all its children are solved
- An action node is closed if at least one of its children is closed
- A non-leaf state node is solved if one of its children is solved
- A non-leaf state node is closed if all its children are closed



Labeling an AND/OR Tree

- An action node is solved if all its children are solved
- An action node is closed if at least one of its children is closed
- A non-leaf state node is solved if one of its children is solved
- A non-leaf state node is closed if all its children are closed

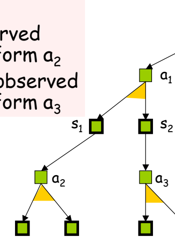


- The problem is solved when the root node is solved
- The problem is impossible if the root node is closed

Solution of an AND/OR Tree

Conditional plan:

- Perform a1
- If s1 is observed then perform a_2
- Else if s_2 is observed then perform a_3



- The solution is the sub-tree that establishes that the root is solved
- It defines a **conditional plan** (or contingency plan) that includes tests on sensory data to pick the next action

Searching an AND/OR Tree

Loop until the root node is solved or closed:

- Top-down generation of the tree:
  Pick a pending state node N that is not solved or closed and expand it (identify all applicable actions and apply them)
- Bottom-up labeling of the tree:
  Update the labeling of the nodes of the tree

[Possibility of expanding state nodes incrementally, one action at a time]

OR Sub-Tree

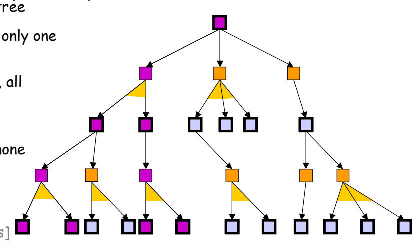
An OR sub-tree corresponds to a path in a classical search tree

For each state node, only one child is included

For each action node, all children are included

It forms a part of a potential solution if none of its nodes is closed

[A solution is an OR sub-tree in which all leaves are goal states]



Best-First Search

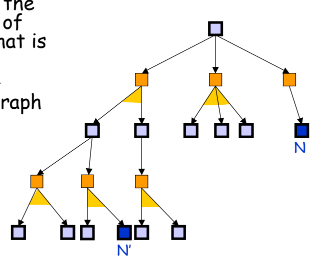
- For every OR sub-tree T in the current AND/OR tree, a best-first search algorithm estimates the cost of the best solution sub-tree containing T and expands a pending state node of the OR sub-tree with the smallest estimated cost
- An algorithm similar to A* – AO*– is available for AND/OR trees

Dealing with Repeated States

- Solution #1:
  Do not test for repeated states
  → Duplicated sub-trees
  [The tree may grow arbitrarily large even if the state space is finite]
- Solution #2:
  Test for repeated states and avoid expanding nodes with repeated states

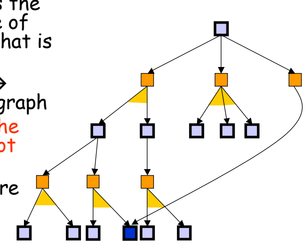
Solution #2 – Case #1

- The state of a newly created node N is the same as the state of another node N' that is not a parent of N
- Merge N and N' → acyclic AND/OR graph



Solution #2 – Case #1

- The state of a newly created node N is the same as the state of another node N' that is not a parent of N
- Merge N and N' → acyclic AND/OR graph
- Just discarding the new node would not work!
- → It makes it more difficult to extract OR sub-trees and manage evaluation function

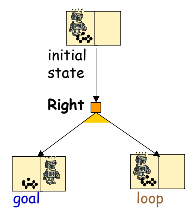


Solution #2 – Case #2

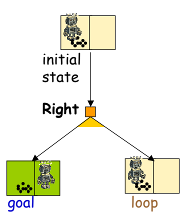
- The state of a newly created node N is the same as the state of a parent of N
- Two possible choices:
  1) Mark N unsolved
  2) Mark N solved
- In either case, the search tree will remain finite, if the state space is finite
- If N is marked solved, the conditional plan may include loops

What does this mean?

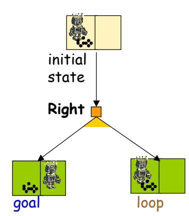
Example



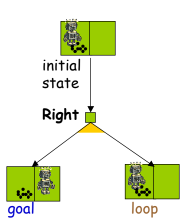
Example



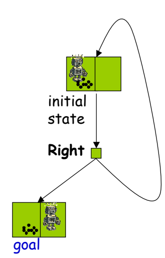
Example



Example



Example



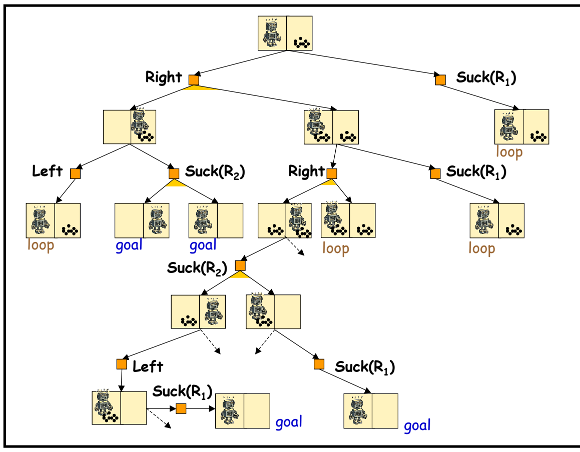
→ cyclic plan:

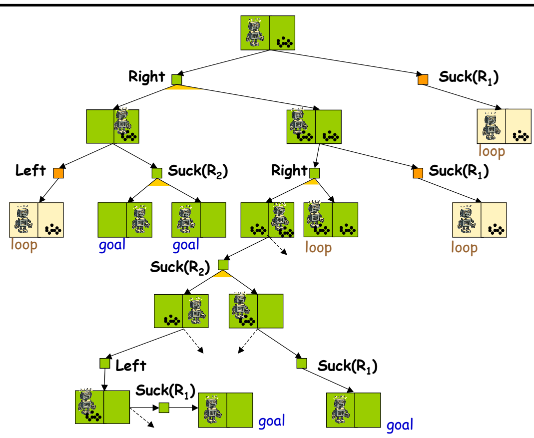
While In(R_1) do **Right**

This plan requires that whenever **Right** is executed, there is a non-zero probability that it does the right thing

The plan is guaranteed only in a probabilistic sense: the probability that it achieves the goal goes to 1 with time, but the running time is not bounded

- In the presence of uncertainty, it's often the case that things don't work the first time as one would like them to work; one must try again
- Without allowing cyclic plans, many problems would have no solution
- So, dealing properly with repeated states in case #2 is much more than just a matter of search efficiency!





Does this always work?

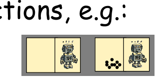
- No ! We must be more careful
- For a cyclic plan to be correct, it should be possible to reach a goal node from every non-goal node in the plan
- → The node labeling algorithm must be slightly modified [left as an exercise]

Uncertainty in Action and Sensing

[Uncertainty strikes twice]

Belief State

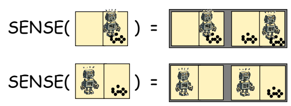
- A belief state is the set of all states that an agent think are possible at any given time or at any stage of planning a course of actions, e.g.:



- To plan a course of actions, the agent searches a space of belief states, instead of a space of states

Sensor Model

- State space S
- The sensor model is a function
  SENSE: S → 2^S
  that maps each state s ∈ S to a belief state (the set of all states that the agent would think possible if it were actually observing state s)
- Example: Assume our vacuum robot can perfectly sense the room it is in and if there is dust in it. But it can't sense if there is dust in the other room

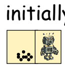


Vacuum Robot Action and Sensor Model

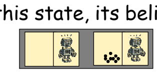
Right P = In(R_1) {E_1 = [D_1 = In(R_1) A_1 = In(R_2)] E_2 = [D_2 = ∅ A_2 = ∅]} [**Right** either does the right thing, or nothing]	**Left** P = In(R_2) {E_1 = [D_1 = In(R_2) A_1 = In(R_1)] E_2 = [D_2 = In(R_2), Clean(R_2) A_2 = In(R_1)]} [**Left** always move the robot to R_1, but it may occasionally deposit dust in R_2]
Suck(r) P = In(r) {E_1 = [D_1 = ∅ A_1 = Clean(r)]} [**Suck** always does the right thing]	• The robot perfectly senses the room it is in and whether there is dust in it • But it can't sense if there is dust in the other room

Transition Between Belief States

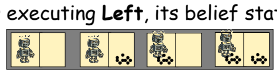
- Suppose the robot is initially in state:



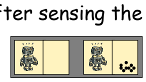
- After sensing this state, its belief state is:



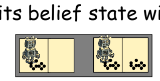
- Just after executing **Left**, its belief state will be:



- After sensing the new state, its belief state will be:



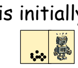
or



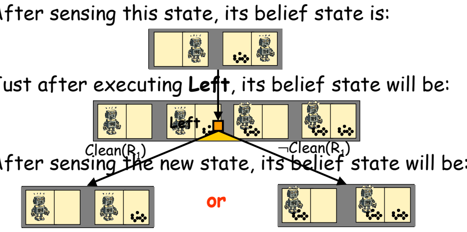
if there is no dust in R_1 — if there is dust in R_1

Transition Between Belief States

- Suppose the robot is initially in state:



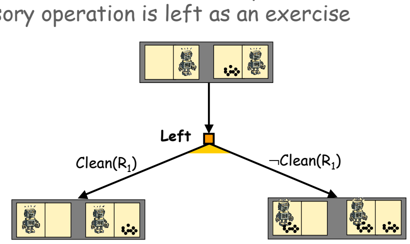
- After sensing this state, its belief state is:
- Just after executing **Left**, its belief state will be:
- After sensing the new state, its belief state will be:



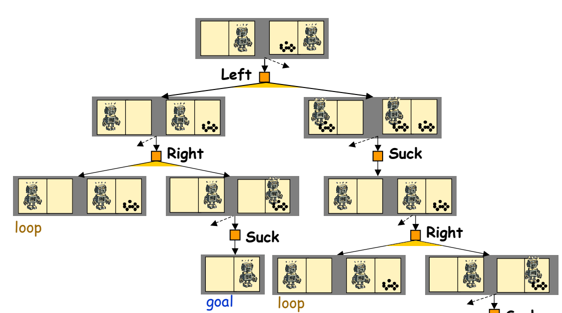
if there is no dust in R_1 — if there is dust in R_1

Transition Between Belief States

A general algorithm for computing the forward projection of a belief state by a combined action-sensory operation is left as an exercise

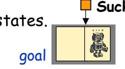


AND/OR Tree of Belief States

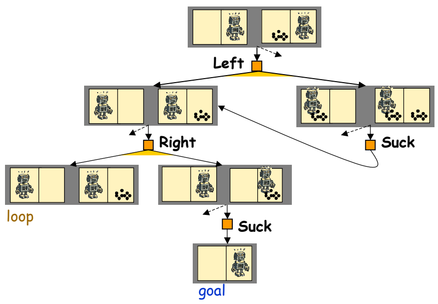


A goal belief state is one in which all states are goal states.

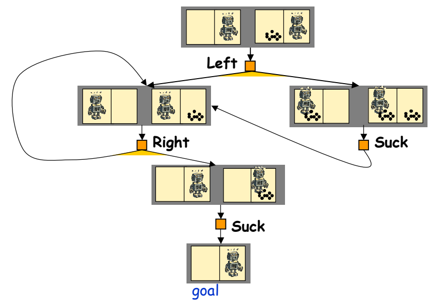
Similarly, an action is applicable to a belief state B if its precondition is achieved in all states in B



AND/OR Tree of Belief States



AND/OR Tree of Belief States



Belief State Representation

Solution #1:

- Represent the set of states explicitly
- If states are described with n propositions, there are $O(2^n)$ states
- The number of belief states is $O(2^{2^n})$
- A belief state may contain $O(2^n)$ states
- This can be hugely expensive

Belief State Representation

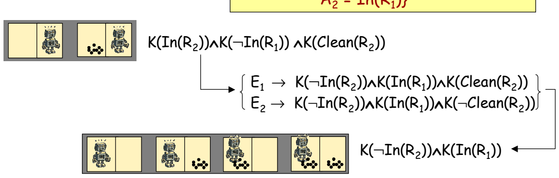
Solution #2:

- Represent only what is known
- For example, if the vacuum robot knows that it is in R_1 (so, not in R_2) and R_2 is clean, then the representation is
  $K(In(R_1)) \wedge K(\neg In(R_2)) \wedge K(Clean(R_2))$
  where K stands for "Knows that ..."
- How many belief states can be represented?
- Only 3^n, instead of $O(2^{2^n})$

Successor of a Belief State Through an Action

An action does not depend on the agent's belief state → K does not appear in the action description (different from R&N, p. 440)

Left
$P = In(R_2)$
$\{E_1 = [D_1 = In(R_2)$
$A_1 = In(R_1)]$
$E_2 = [D_2 = In(R_2), Clean(R_2)$
$A_2 = In(R_1)]\}$



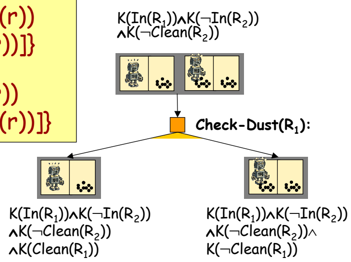
Sensory Actions

- So far, we have assumed a unique sensory operation automatically performed after executing of each action of a plan
- But an agent may have several sensors, each having some cost (e.g., time) to use
- In certain situations, the agent may like better to avoid the cost of using a sensor, even if using the sensor could reduce uncertainty
- This leads to introducing specific sensory actions, each with its own representation → active sensing
- Like with other actions, the agent chooses which sensory actions it want to execute and when

Example

Check-Dust(r):
P = In(Robot,r)
{when Clean(r)
D = K(¬Clean(r))
A = K(Clean(r))]}
{when ¬Clean(r)
D = K(Clean(r))
A = K(¬Clean(r))]}

A sensory action maps a state into a belief state
Its precondition is about the state
Its effects are on the belief state



Precondition Issue

- In complex worlds, actions may have long preconditions, e.g.:

  Drive-Car:
  P = Have(Keys) ∧ ¬Empty(Gas-Tank) ∧ Battery-Ok ∧ Ignition-Ok ∧ ¬Flat-Tires ∧ ¬Stolen(Car) ∧ ...
- In the presence of non-deterministic uncertainty, few actions, if any, will be applicable to a belief state
- → Use of default information

Default Rules

Example:

- If a belief state contains
  K(Bird(Tweety))
  but does not contain
  K(Flies(Tweety)) or K(¬Flies(Tweety))
  [the agent does not know if Tweety can fly or not]
  then assume
  K(Flies(Tweety))
  and include it in the belief state
- If later the agent observes that Tweety doesn't fly – e.g., is a penguin – then K(Flies(Tweety)) will be retracted and replaced by K(¬Flies(Tweety))

Application to Action Preconditions

- The precondition of **Drive-Car**:
  Have(Keys) ∧ ¬Empty(Gas-Tank) ∧ Battery-Ok ∧ SparkPlugs-Ok ∧ ¬Flat-Tires ∧ ¬Stolen(Car) ...
  is replaced by:
  Have(Keys) ∧ Normal(Car)
- The following state constraints are added to define Normal(Car):
  Empty(Gas-Tank) → ¬Normal(Car)
  ¬Battery-Ok → ¬Normal(Car)
  ¬SparkPlugs-Ok → ¬Normal(Car)
- The default rule is:
  Unless K(¬Normal(Car)) is in the belief state, assume K(Normal(Car))

- If executing Drive-Car fails to produce the expected effects, then the agent should consider the conditions in the left-hand sides of the state constraints defining ¬Normal(Car) as prime suspects and check (i.e., sense) them
- Unfortunately, it is quite difficult to manage default information appropriately [see R&N: Chap. 10, Sect. 10.7]