

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:

$$P$$
$$\{E_1, E_2, \dots, E_r\}$$

where each E_i , $i = 1, \dots, r$ describes one possible set of effects of the action in a state satisfying P

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

Example: Devious Vacuum Robot

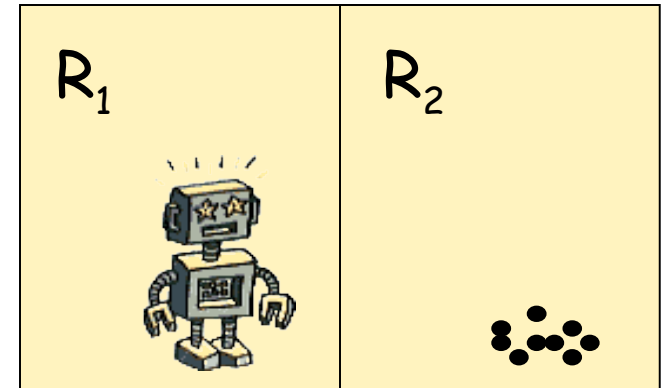
Right

$$P = \text{In}(R_1)$$

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

$$E_2 = [D_2 = \text{Clean}(R_1) \\ A_2 = \emptyset]\}$$

Not intentional,
so unpredictable ←



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

Left

$$P = \text{In}(R_2)$$

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

Left always leads the robot to move to R_1

Suck(R_1)

$P = \text{In}(R_1)$

$\{E_1 = [D_1 = \emptyset$

$A_1 = \text{Clean}(R_1)]\}$

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

Suck(R_2)

$P = \text{In}(R_2)$

$\{E_1 = [D_1 = \emptyset$

$A_1 = \text{Clean}(R_2)]$

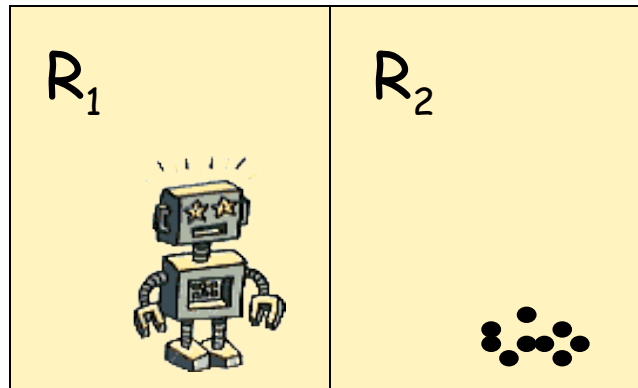
$E_2 = [D_2 = \text{In}(R_2)$

$A_2 = \text{Clean}(R_2), \text{In}(R_1)]\}$

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

Problem

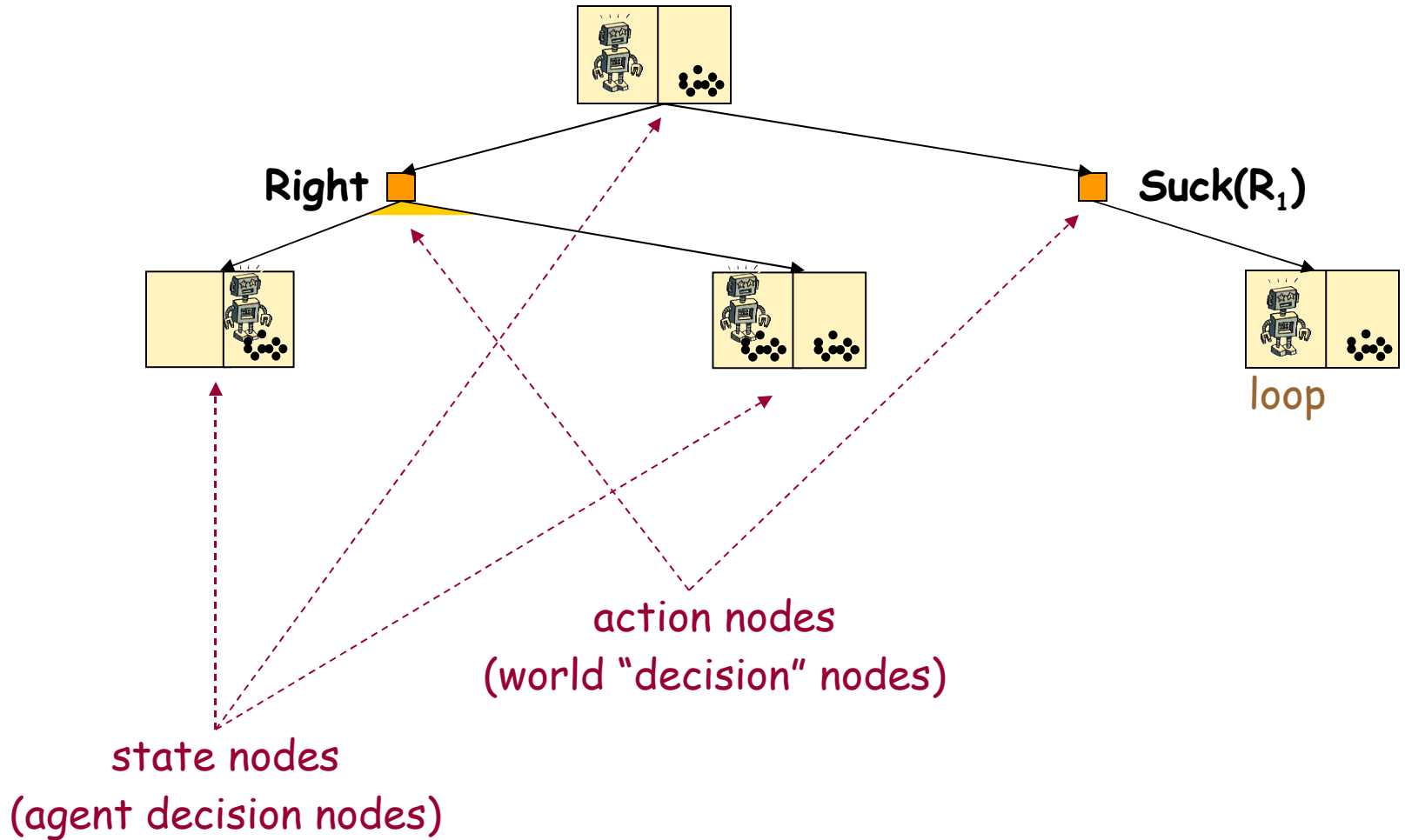
From the initial state:



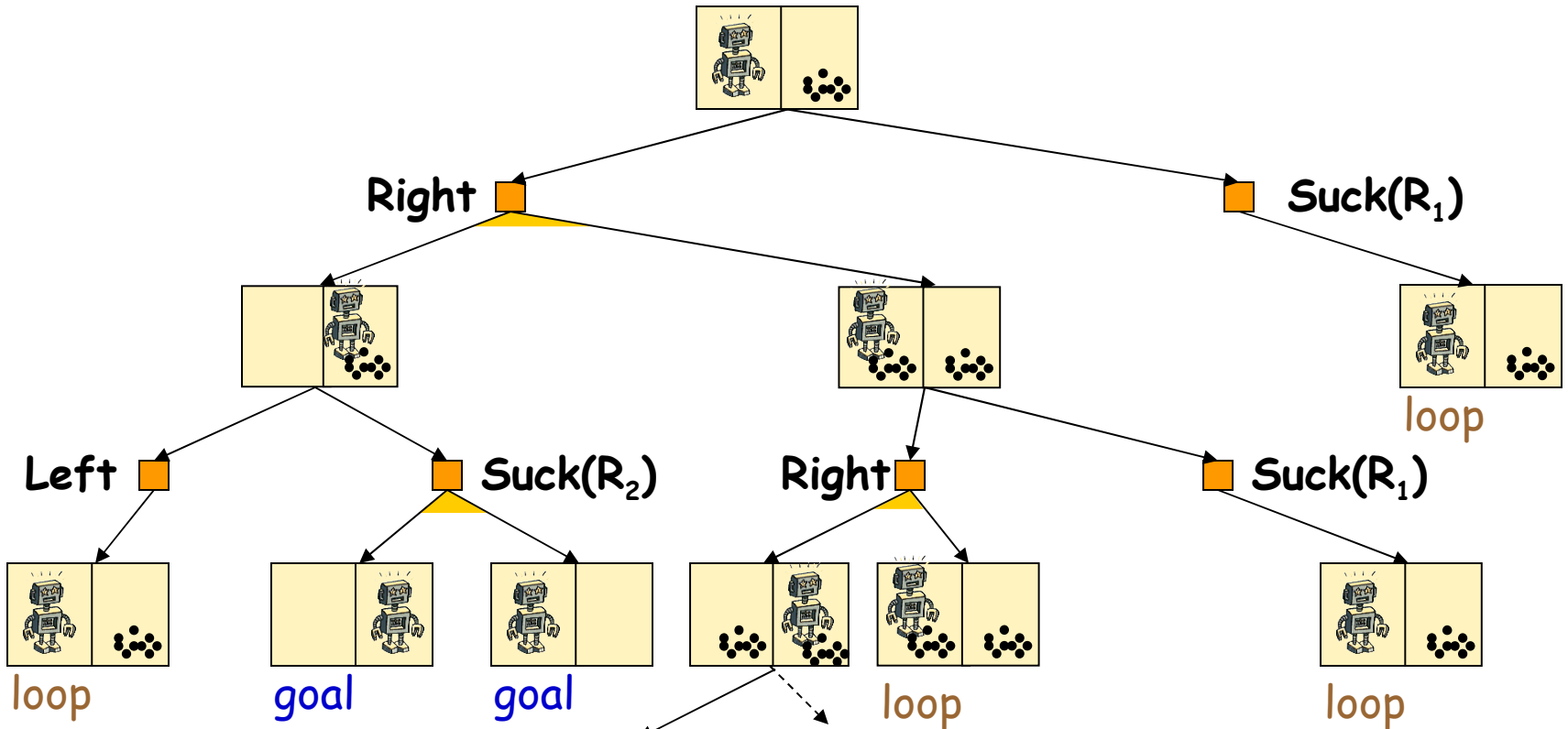
our devious vacuum robot must achieve the goal
 $\text{Clean}(R_1) \wedge \text{Clean}(R_2)$

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

AND/OR Tree



AND/OR Tree



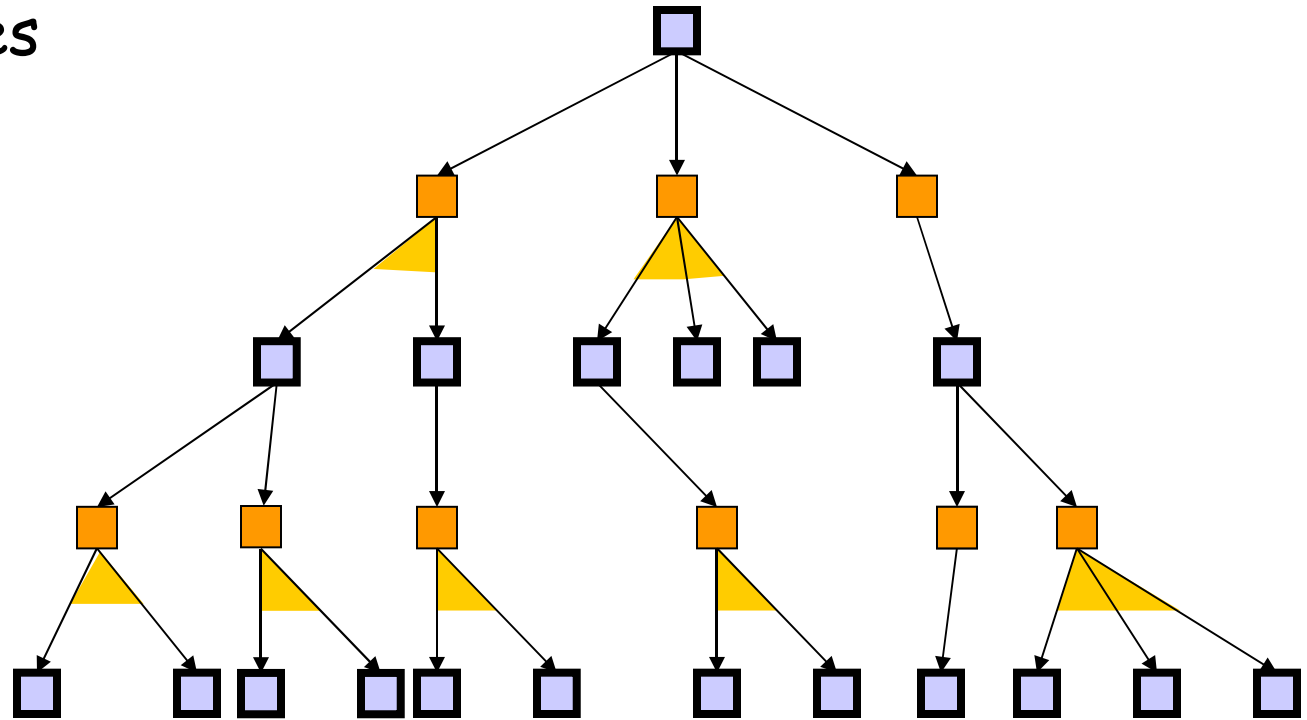
This is essentially forward planning

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

[not as obvious as it looks]

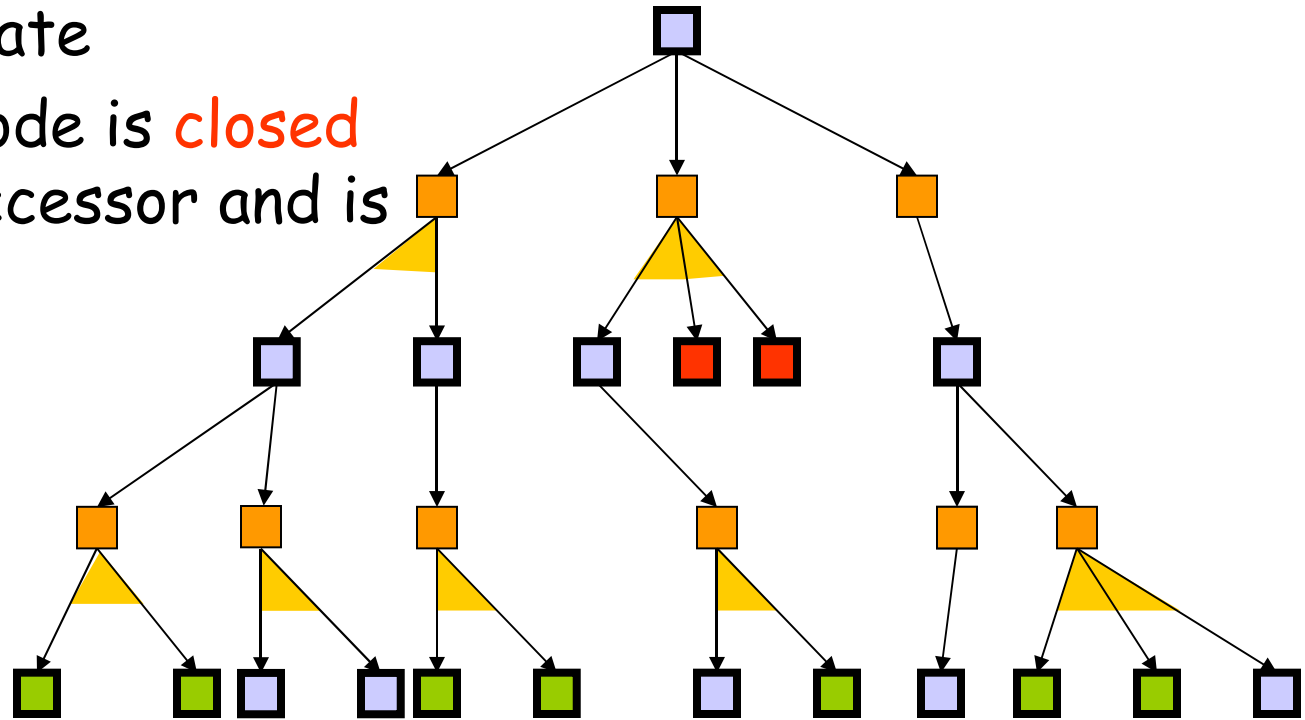
Labeling an AND/OR Tree

- Assume no detection of revisited states



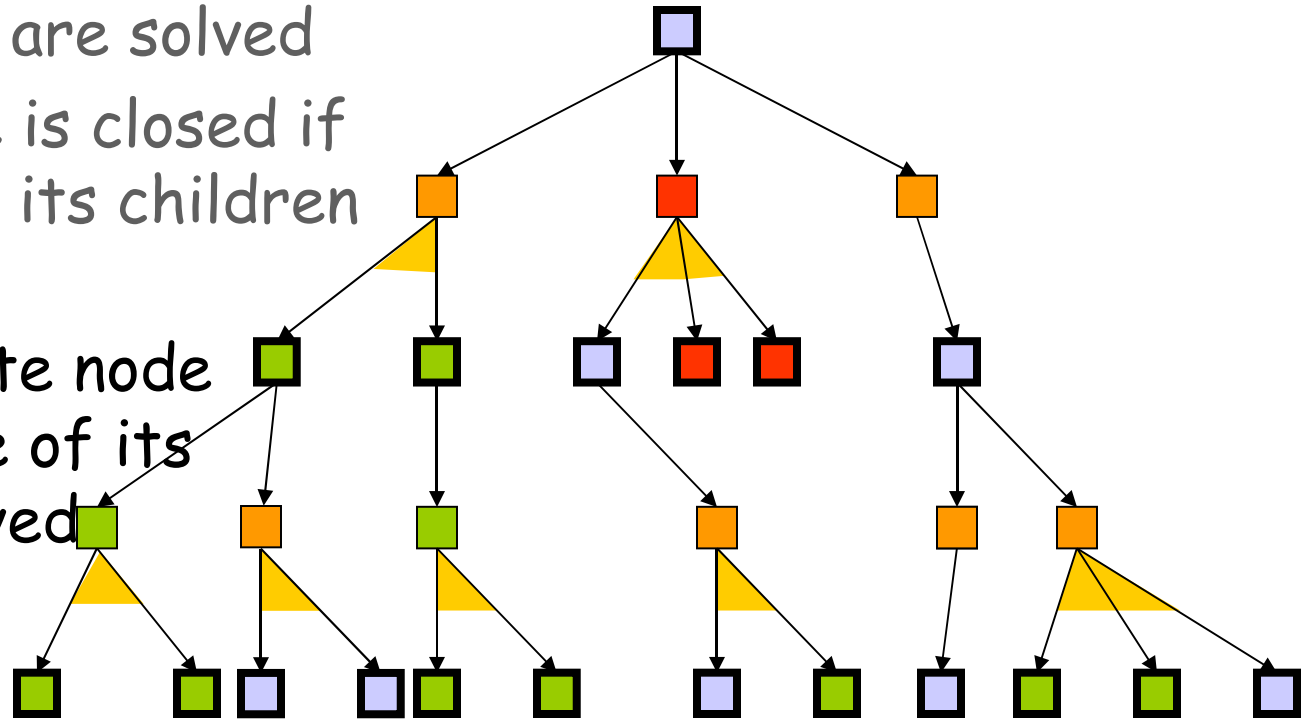
Labeling an AND/OR Tree

- 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
- 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**

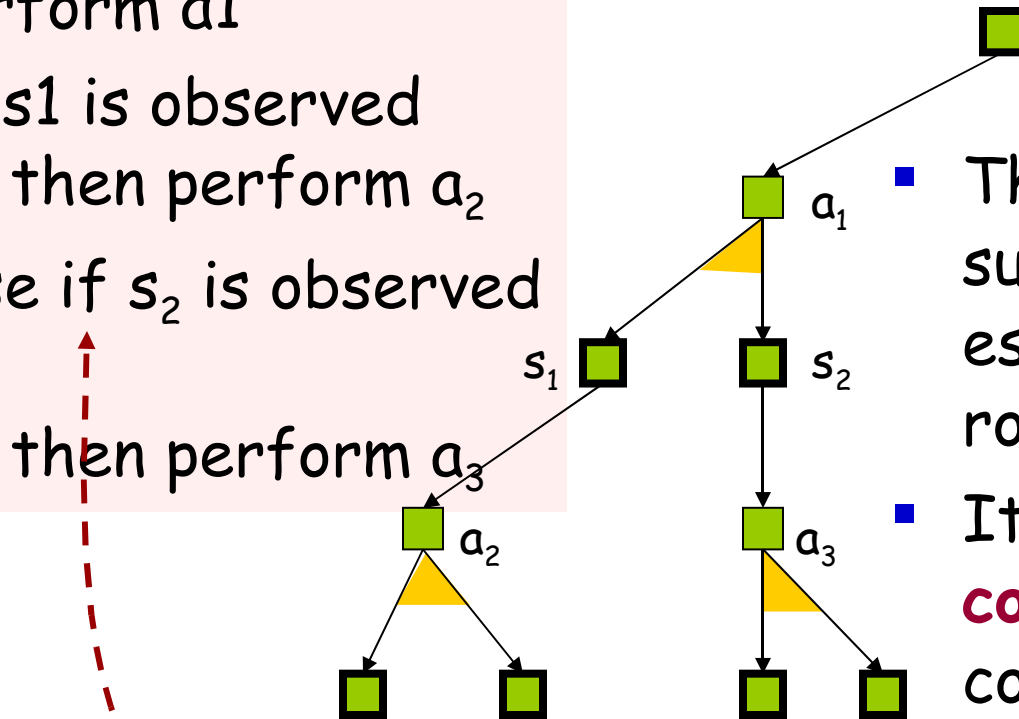


Solution of an AND/OR Tree

Conditional plan:

- Perform a_1
- If s_1 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:**

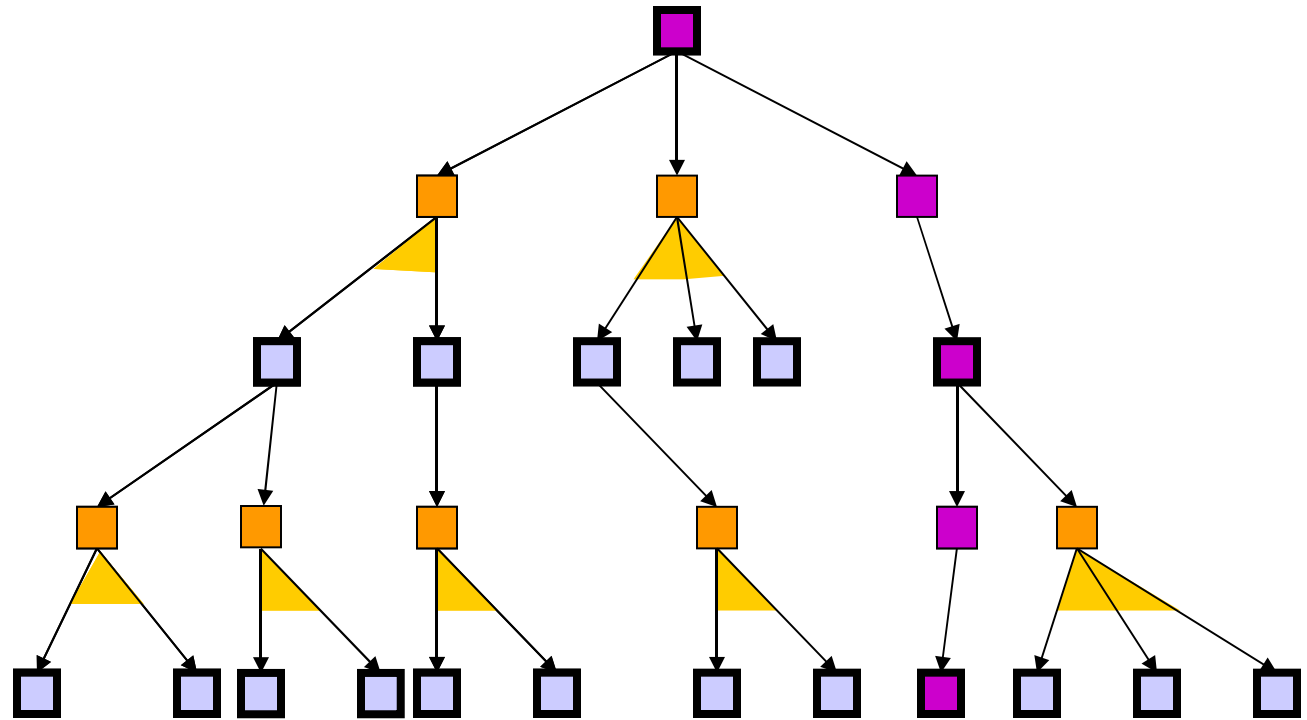
Pick a pending state node N that is not solved or closed and expand it (identify all applicable actions and apply them)

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

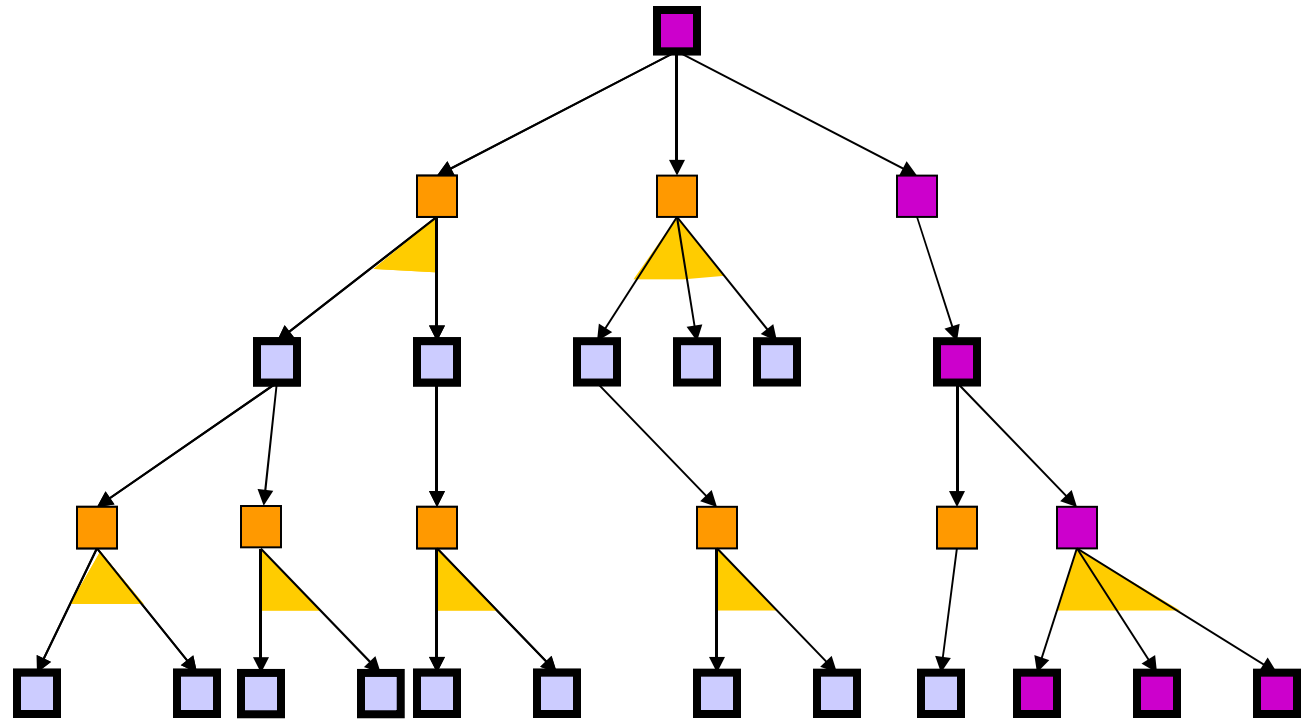
- **Bottom-up labeling of the tree:**

Update the labeling of the nodes of the tree

Another OR Sub-Tree



Another OR Sub-Tree



Best-First Search

- Let T be any OR sub-tree in the current AND/OR tree and f be a function that estimates the cost of the best solution sub-tree containing T ,

$$\text{e.g., } f(T) = g(T) + h(T)$$

where $g(T)$ is the cost of T and $h(T)$ is an estimate of the cost from T to a solution sub-tree.

- Best-first search 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 Revisited States

- **Solution #1:**

Do not test for revisited states

→ Duplicated sub-trees

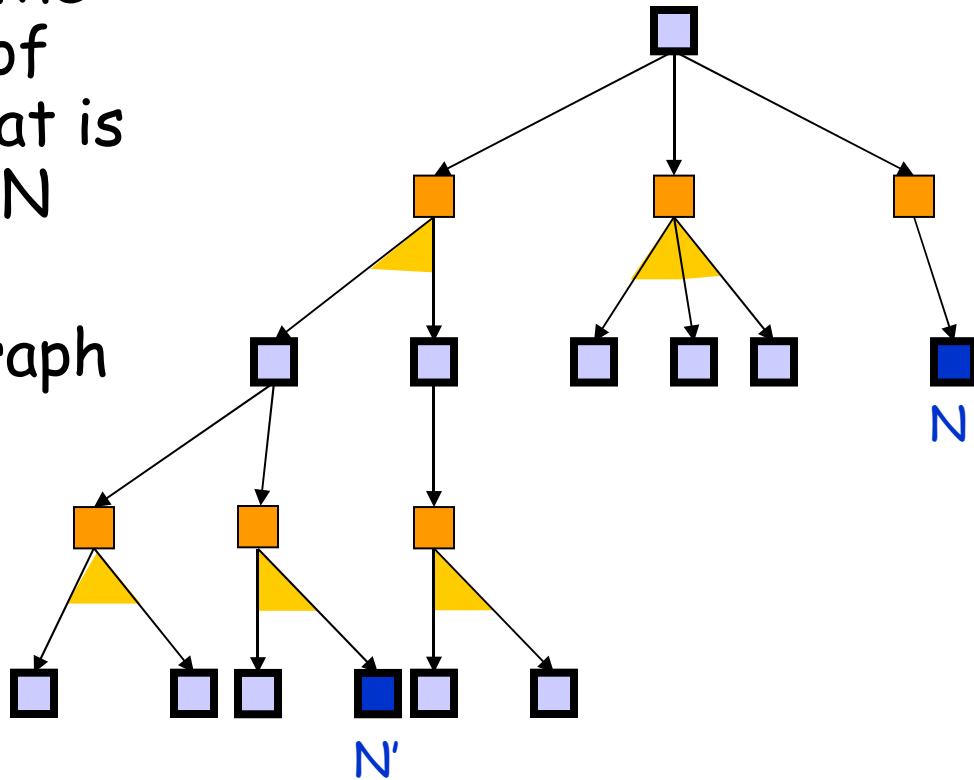
[The tree may grow arbitrarily large even if the state space is finite]

- **Solution #2:**

Test for revisited states and avoid expanding nodes with revisited states

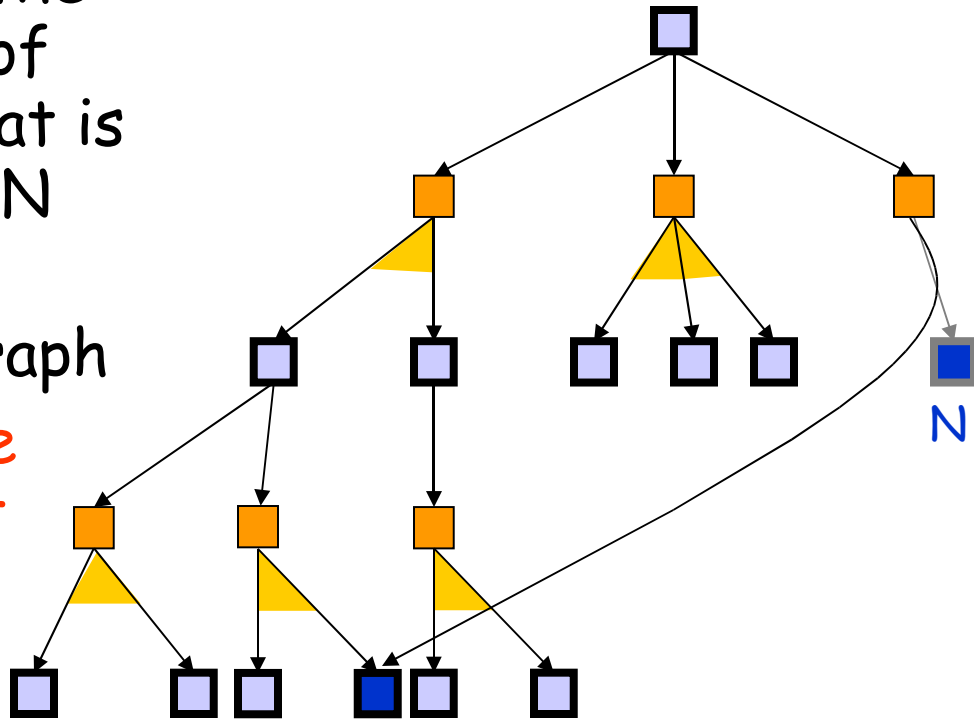
Solution #2 - Case #1

- The state of a newly created node N is the same as the state of another node N' that is not an ancestor of N
- Merge N and $N' \rightarrow$ 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 an ancestor of N
- Merge N and $N' \rightarrow$ acyclic AND/OR graph
- Just discarding the new node would not work! Why??
- This makes it more difficult to extract OR sub-trees and merge

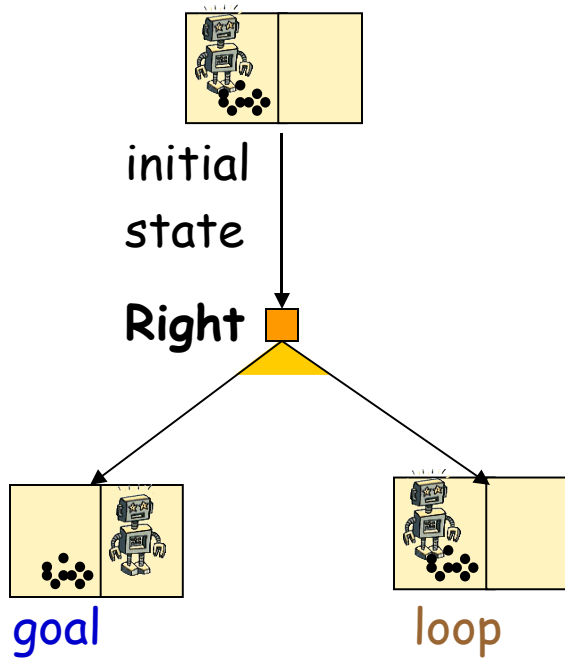


Solution #2 - Case #2

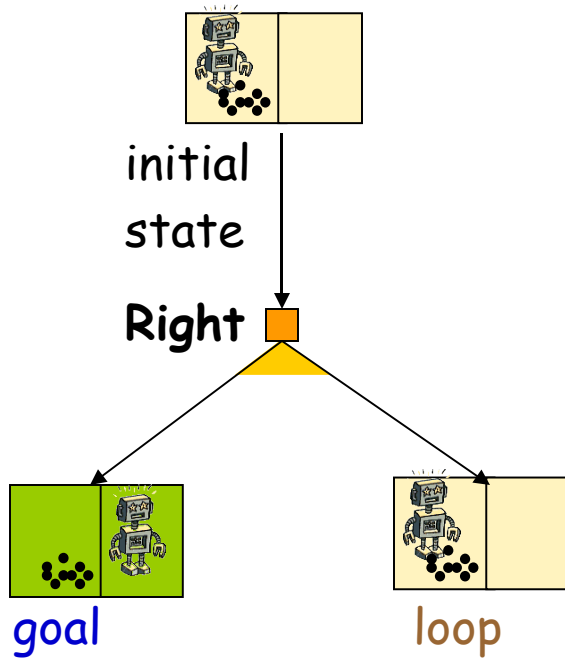
- The state of a newly created node N is the same as the state of a parent of N
- Two possible choices:
 - Mark N **closed**
 - Mark N **solved**
- In either case, the search tree will remain finite, if the state space is finite
- If N is marked solved, the conditional plan may include loops

What does this mean???

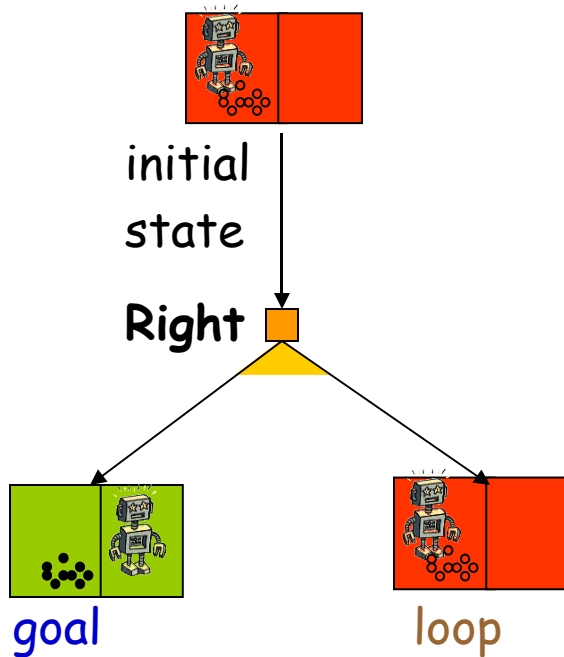
Example



Example

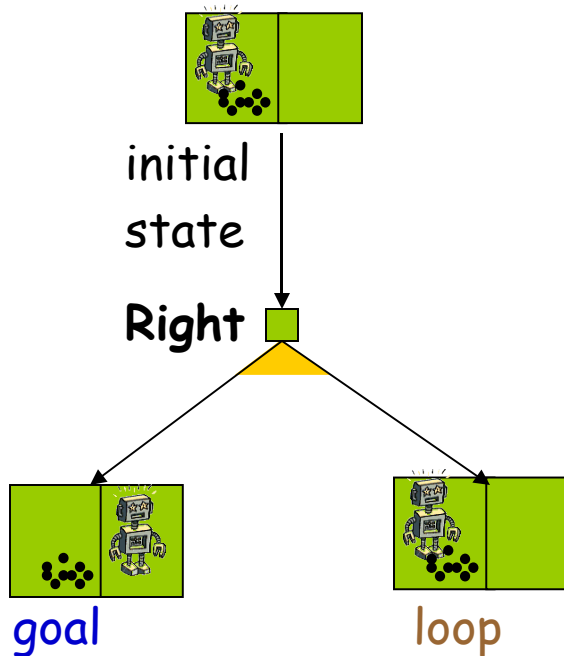


Example



Marking loop nodes closed
→ The problem has no solution

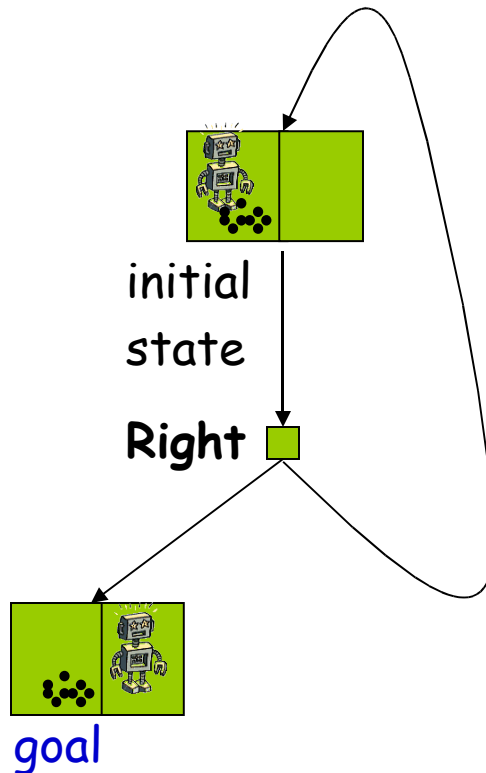
Example



Marking loop nodes closed
→ The problem has a solution

But what is this solution?

Example



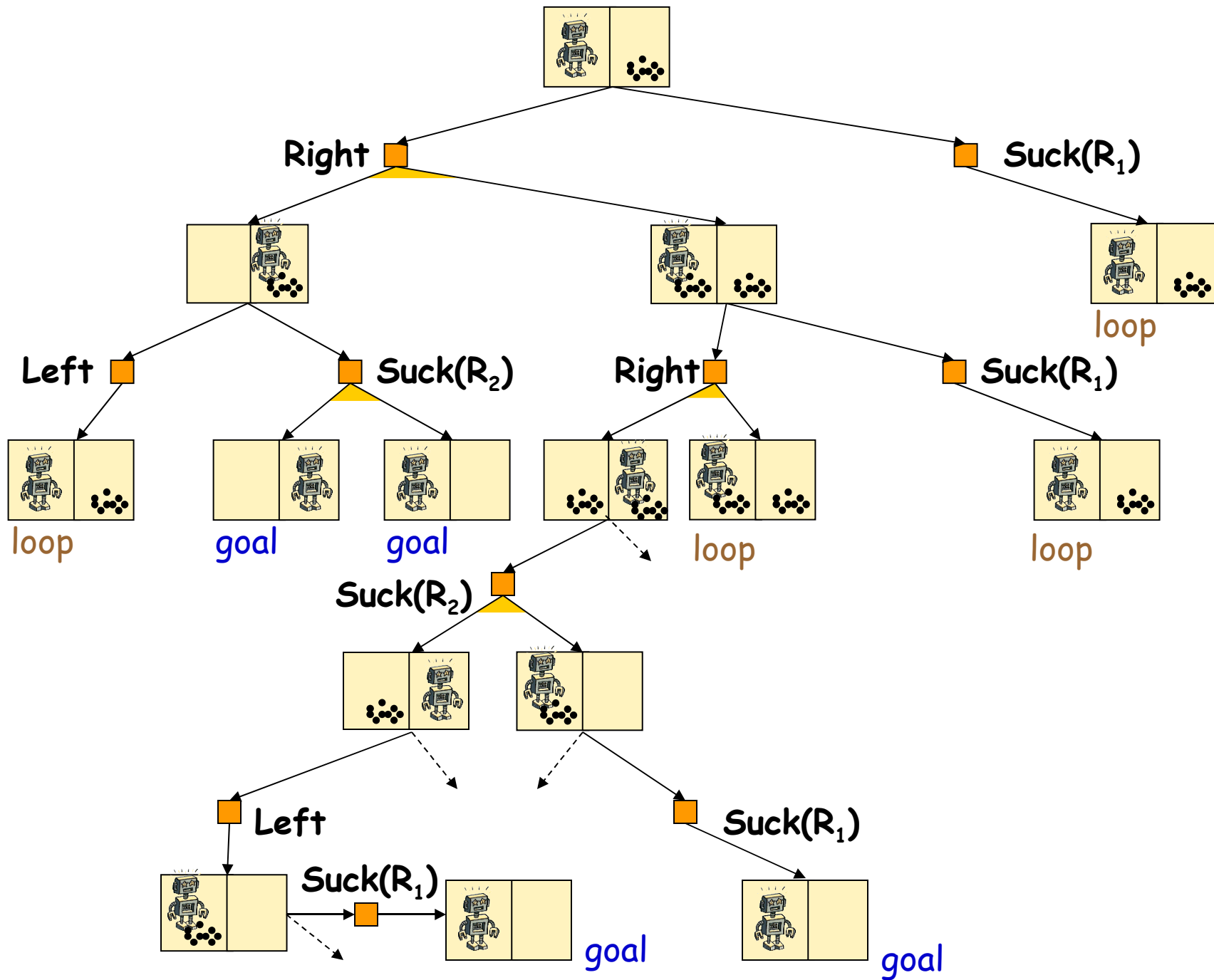
→ **cyclic** plan:

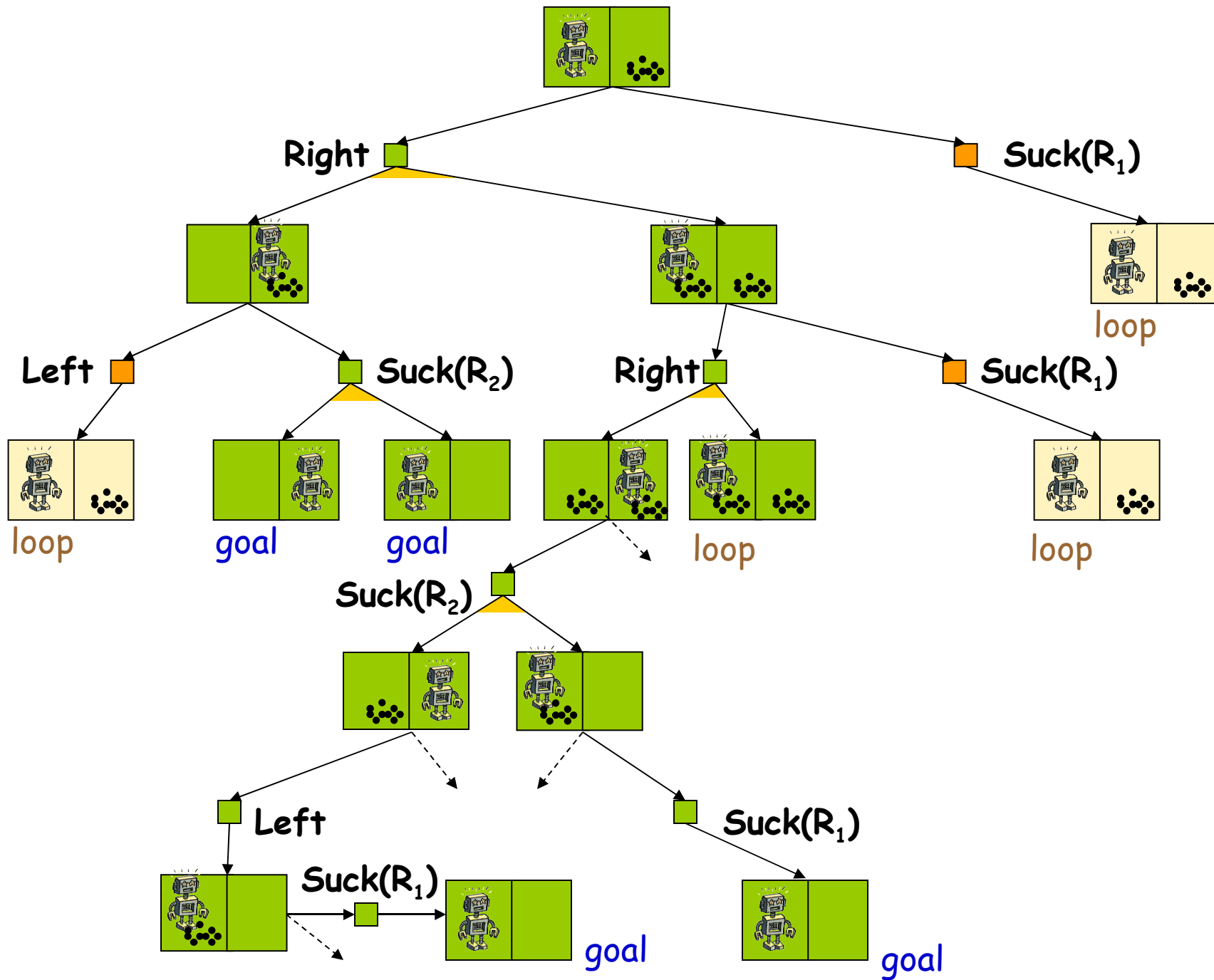
While In(R_1) do Right

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

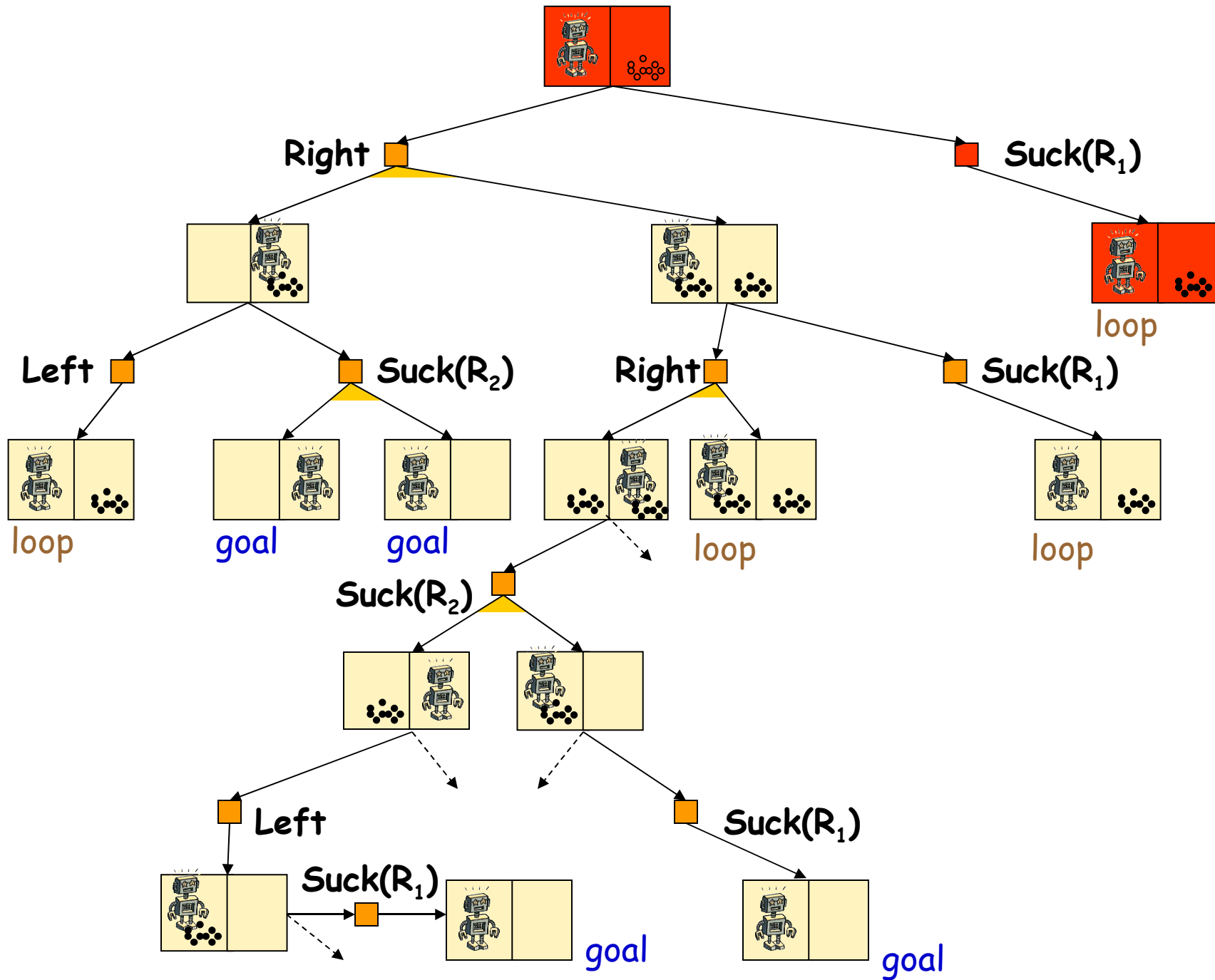
- In the presence of uncertainty, it's often the case that things don't work the first time as one would like them to work; one must try again
- Without allowing cyclic plans, many problems would have no solution
- So, dealing properly with repeated states in case #2 is much more than just a matter of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



Does this always work?

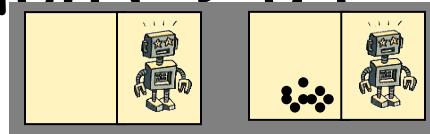
- No ! We must be more careful
- For a cyclic plan to be correct, it should be possible to reach a goal node from every non-goal node in the plan
- → The node labeling algorithm must be slightly modified [left as an exercise]

Uncertainty in Action and Sensing

[Uncertainty strikes twice]

Belief State

- A **belief state** is the set of all states that an agent thinks 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

$$\text{SENSE}: S \rightarrow 2^S$$

that maps each state $s \in S$ 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

$$\text{SENSE} \left(\begin{array}{|c|c|} \hline & \text{robot} \\ \hline \end{array} \right) = \begin{array}{|c|c|c|c|} \hline & \text{robot} & & \text{robot} \\ \hline & \text{dust} & \text{dust} & \text{dust} \\ \hline \end{array}$$

$$\text{SENSE} \left(\begin{array}{|c|c|} \hline \text{robot} & \text{dust} \\ \hline \end{array} \right) = \begin{array}{|c|c|c|c|} \hline \text{robot} & & \text{robot} & \text{dust} \\ \hline \end{array}$$

Vacuum Robot Action and Sensor Model

Right

$$P = \text{In}(R_1)$$

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

$$E_2 = [D_2 = \emptyset \\ A_2 = \emptyset]\}$$

[Right either does the right thing, or nothing]

Suck(r)

$$P = \text{In}(r)$$

$$\{E_1 = [D_1 = \emptyset \\ A_1 = \text{Clean}(r)]\}$$

[Suck always does the right thing]

Left

$$P = \text{In}(R_2)$$

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

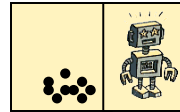
$$E_2 = [D_2 = \text{In}(R_2), \text{Clean}(R_2) \\ A_2 = \text{In}(R_1)]\}$$

[Left always move the robot to R_1 , but it may occasionally deposit dust in R_2]

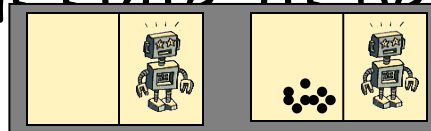
- The robot perfectly senses the room it is in and whether there is dust in it
- But it can't sense if there is dust in the other room

Transition Between Belief States

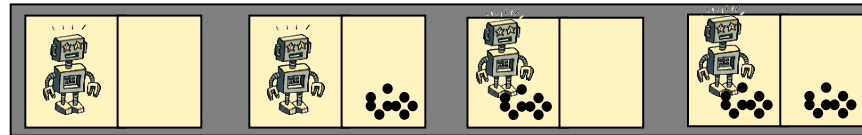
- Suppose the robot is initially in state:



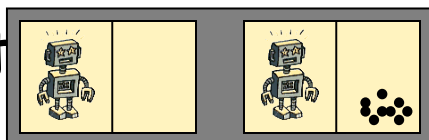
- After sensing this state, its belief state is:



- Just after the state will be:



- After a new state, its belief state will be:



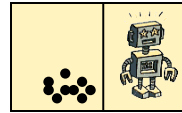
or

if there is no dust

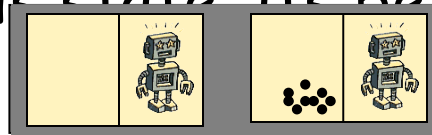
if there is dust in R_1

Transition Between Belief States

- Suppose the robot is initially in state:



- After sensing this state, its belief state is:



- Just after state will be:

Clean(R_1)

\neg Clean(R_1)

- After new state, its will be:

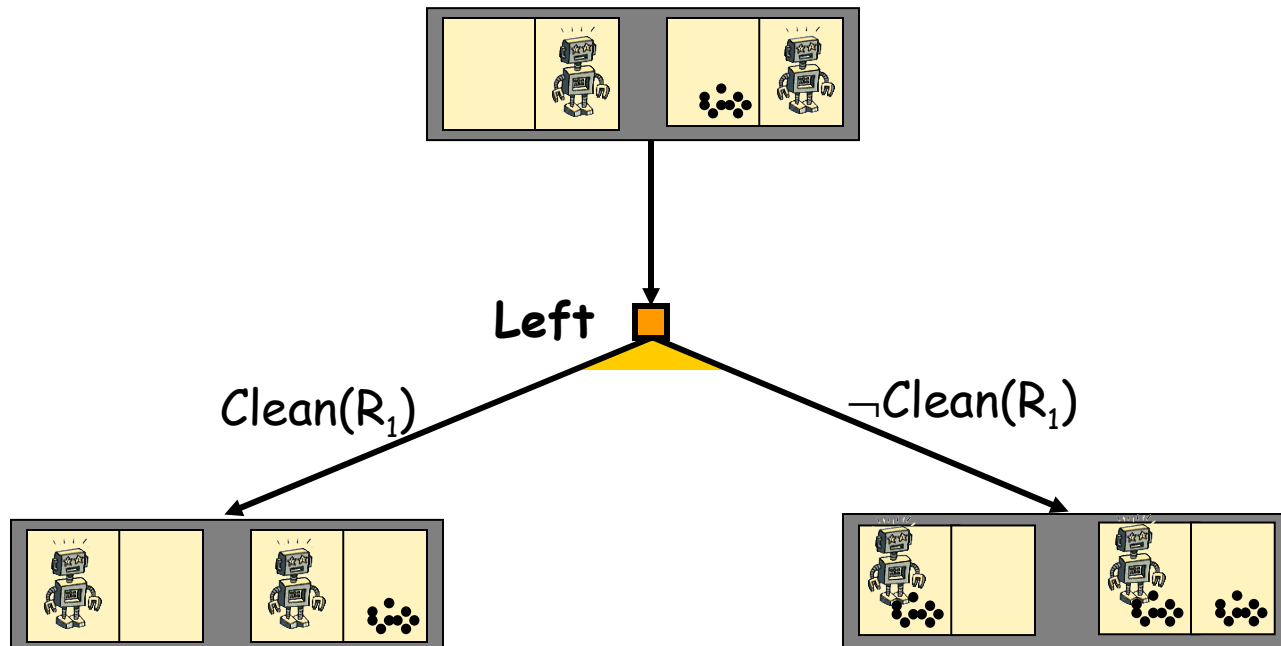
or

if there is no dust

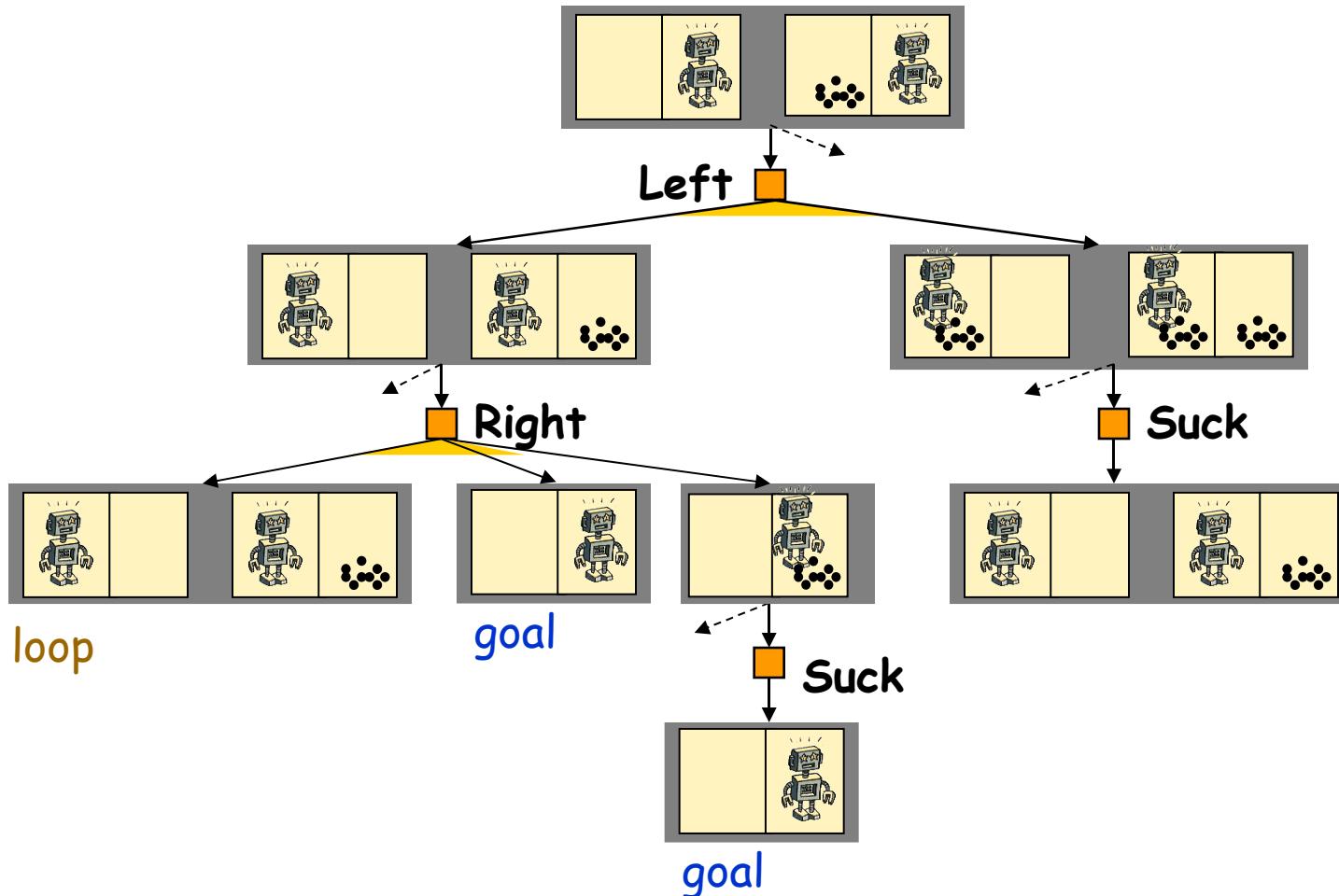
if there is dust in R_1

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



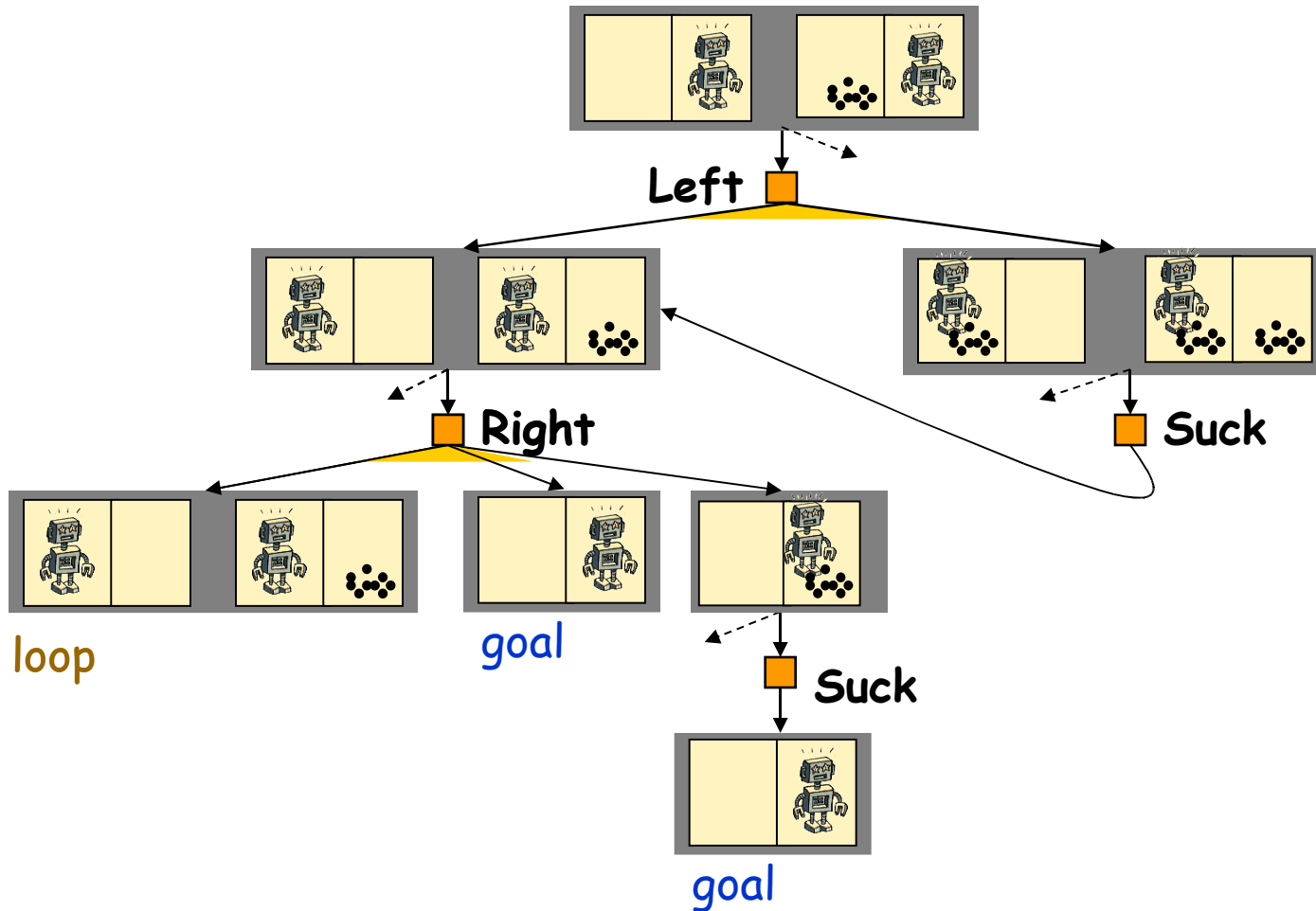
AND/OR Tree of Belief States



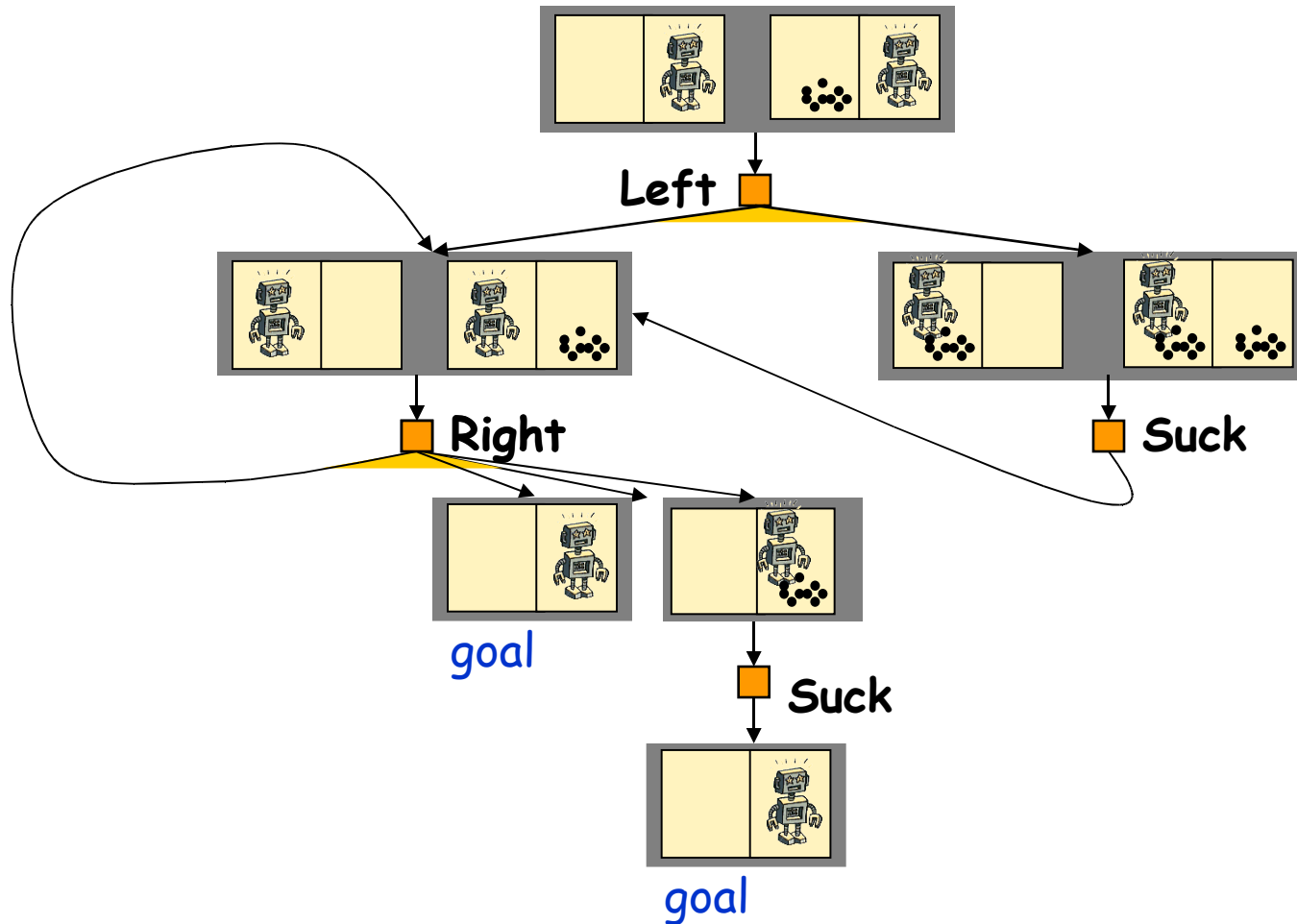
An action is applicable to a belief state B if its precondition is achieved in all states in B

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

AND/OR Tree of Belief States



AND/OR Tree of Belief States



Belief State Representation

Solution #1:

- Represent the set of states explicitly
- Under the closed world assumption, 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

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

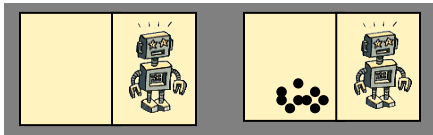
$$K(\text{In}(R_1)) \wedge K(\neg \text{In}(R_2)) \wedge K(\text{Clean}(R_2))$$

where K stands for "Knows that ..."

- How many belief states $O(2^{2^n})$ can be represented?
- Only 3^n instead of

Successor of a Belief State Through an Action

An action does not depend on the agent's belief state
 \rightarrow K does not appear in the action description
 the action description



$$K(\text{In}(R_2)) \wedge K(\neg \text{In}(R_1)) \wedge K(\text{Clean}(R_2))$$

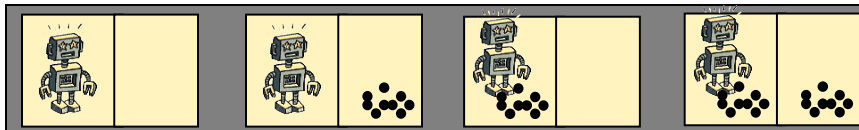
Left

$P = \text{In}(R_2)$

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

$E_2 = [D_2 = \text{In}(R_2), \text{Clean}(R_2)$
 $A_2 = \text{In}(R_1)]\}$

$$\left\{ \begin{array}{l} E_1 \rightarrow K(\neg \text{In}(R_2)) \wedge K(\text{In}(R_1)) \wedge K(\text{Clean}(R_2)) \\ E_2 \rightarrow K(\neg \text{In}(R_2)) \wedge K(\text{In}(R_1)) \wedge K(\neg \text{Clean}(R_2)) \end{array} \right\}$$



$$K(\neg \text{In}(R_2)) \wedge K(\text{In}(R_1))$$

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

Example

Check-Dust(r):

$P = \text{In}(\text{Robot}, r)$

{when Clean(r)

$D = K(\neg \text{Clean}(r))$

$A = K(\text{Clean}(r))$ }]

{when $\neg \text{Clean}(r)$

$D = K(\text{Clean}(r))$

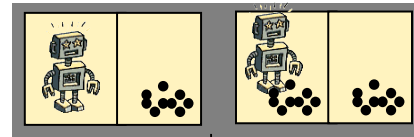
$A = K(\neg \text{Clean}(r))$ }]

A sensory action maps a state into a belief state

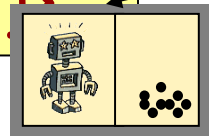
Its precondition is about the state

Its effects are on the belief state

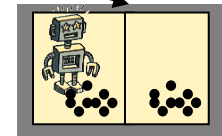
$K(\text{In}(R_1)) \wedge K(\neg \text{In}(R_2))$
 $\wedge K(\neg \text{Clean}(R_2))$



Check-Dust(R_1):



$K(\text{In}(R_1)) \wedge K(\neg \text{In}(R_2))$
 $\wedge K(\neg \text{Clean}(R_2))$
 $\wedge K(\text{Clean}(R_1))$



$K(\text{In}(R_1)) \wedge K(\neg \text{In}(R_2))$
 $\wedge K(\neg \text{Clean}(R_2)) \wedge$
 $K(\neg \text{Clean}(R_1))$

Precondition Issue

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

Drive-Car:

$$P = \text{Have(Keys)} \wedge \neg \text{Empty(Gas-Tank)} \wedge \text{Battery-Ok} \\ \wedge \text{Ignition-Ok} \wedge \neg \text{Flat-Tires} \wedge \neg \text{Stolen(Car)} \wedge \dots$$

- In the presence of non-deterministic uncertainty, few actions, if any, will be applicable to a belief state
- → Use of default rule

Default Rule

- The precondition of **Drive-Car**:

$\text{Have}(\text{Keys}) \wedge \neg \text{Empty}(\text{Gas-Tank}) \wedge \text{Battery-Ok}$
 $\wedge \text{SparkPlugs-Ok} \wedge \neg \text{Flat-Tires} \wedge \neg \text{Stolen}(\text{Car}) \dots$

is replaced by:

$\text{Have}(\text{Keys}) \wedge \text{Normal}(\text{Car})$

- The following state constraints are added to define **Normal(Car)**:

$\text{Empty}(\text{Gas-Tank}) \rightarrow \neg \text{Normal}(\text{Car})$

$\neg \text{Battery-Ok} \rightarrow \neg \text{Normal}(\text{Car})$

$\neg \text{SparkPlugs-Ok} \rightarrow \neg \text{Normal}(\text{Car})$

- The **default rule** is:

Unless $K(\neg \text{Normal}(\text{Car}))$ is in the belief state assume

- 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 \neg 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]

Conclusion

- We have seen today how to plan with uncertainty, but have said nothing about the uncertainty itself
- Next time, we will look at *probabilistic uncertainty* where we assume that we know something about the likelihood of the different possible outcomes of actions or states.
- Please read Chapter 13 in preparation
- REMINDER: Midterm tonight 7-9 pm, in Gates B12

Good luck with your final studying!

Suggestion

- It's time to refresh your memory on probability theory:
 - axioms of probability
 - random variable
 - joint distributions
 - conditioning
 - independence

[R&N: Chap. 13, Sect. 13.3-6]

- We will be using probabilities in the next lecture

Conclusion

REMINDER: Midterm tonight 7-9 pm, in Gates B12

Good luck with your final studying!