Lecture 8

CS 6380 Artificial Intelligence

Local Search in Continuous and Non-Deterministic Settings

Department of Computer Science and Engineering

Indian Institute of Technology Madras (IITM), India

Some material based on lecture slides for CSE 571: Artificial Intelligence by Siddharth Srivastava, ASU.

Logistics and Recap

Talk Announcement

- **Title:** Towards Reliable Autonomy: From Individual Agents to Autonomous Teams
- **Date:** August 21, 2026
- **Time: 11**:00 AM – 12:00 PM
- Location: SSB 334

Dr. Sandhya Saisubramanian
Oregon State University, USA
https://www.sandhyasai.com/

Office Hours

- When?
  - Monday: 3:15 PM – 4:15 PM
  - Tuesday: 4:45 PM – 5:45 PM

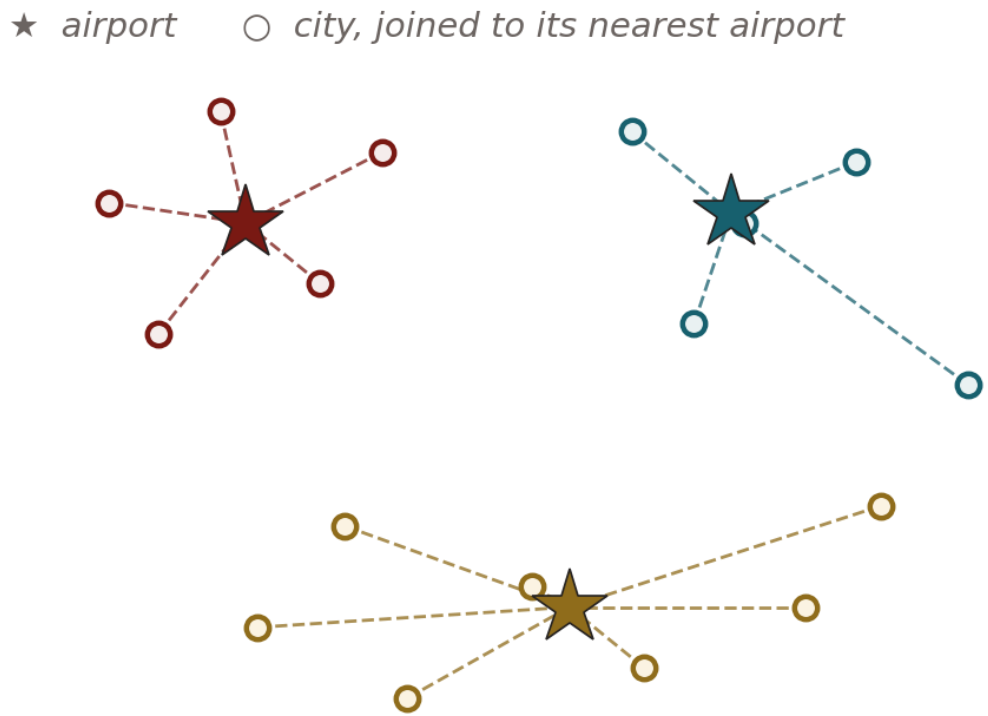
- Where?
  - SSB 304

Local Search in Continuous Spaces

When the state is a vector of real numbers

Why Continuous Spaces Break Our Machinery

- Most real problems are continuous: positions, angles, prices, network weights
- The successor function collapses: a state has infinitely many neighbours, so "generate all successors" means nothing
- Running example: site 3 airports in Romania
  - state $x = (x_1, y_1, x_2, y_2, x_3, y_3)$: six real numbers
  - minimise $f(x) = \Sigma_a \Sigma_{c \in C_a} (x_a - x_c)^2 + (y_a - y_c)^2$
  - C_a = the cities whose nearest airport is a
- *Careful: C_a changes as the airports move, so f is smooth in patches — differentiable almost everywhere, not everywhere*



Three Ways to Cope

1. Discretize it

Fix a step δ and allow only ±δ in each coordinate. A state now has 2n neighbours, and every discrete algorithm from today applies unchanged.

The cost: δ too big and you step over the optimum, too small and you crawl.

2. Perturb at random

Don't enumerate neighbours: sample one. Add a small random vector to the current state and test it.

This is what first-choice hill climbing and simulated annealing already do, so they carry over to continuous spaces with no changes at all.

3. Use the gradient

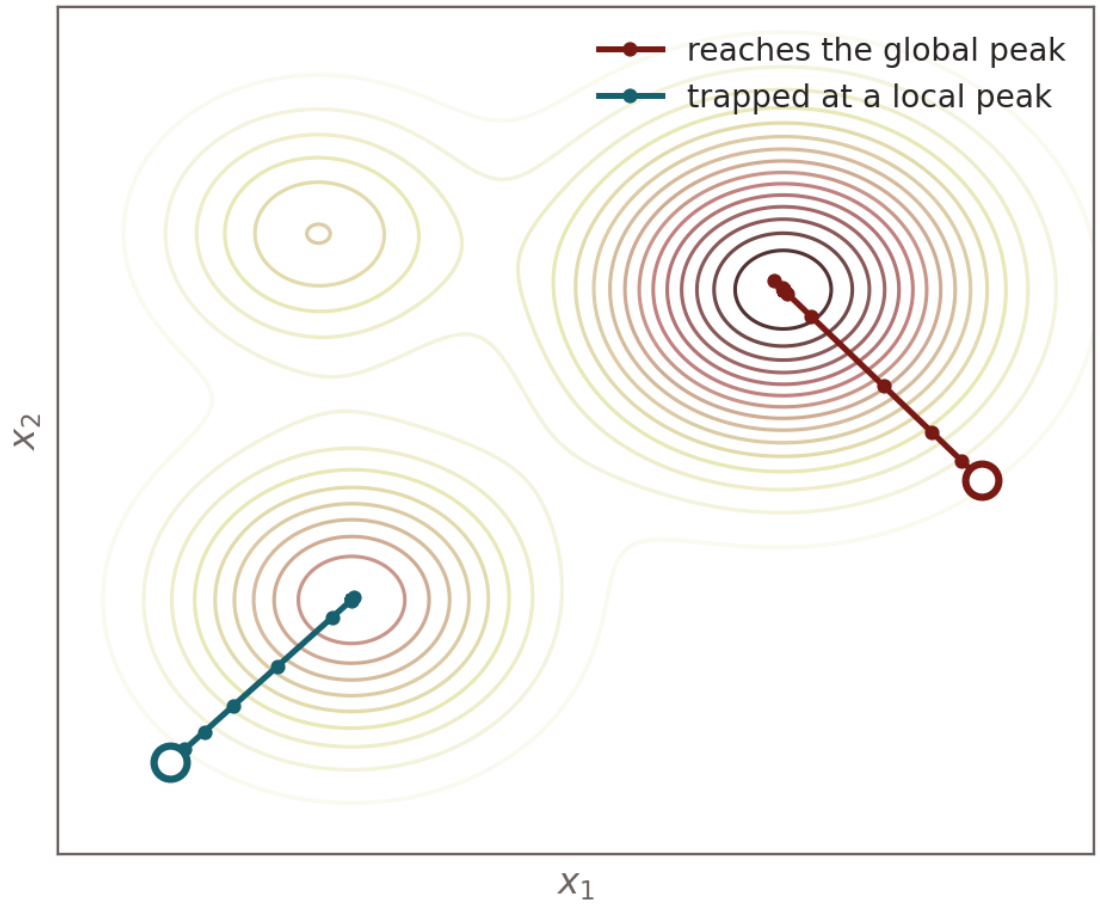
If f can be differentiated, the direction of steepest ascent is available for free: no neighbours needed.

If it cannot, estimate it empirically: evaluate f at a few perturbed points and read off the slope.

Only the third option is new. The first two are ways of pretending the space is discrete so that the algorithms you already have keep working: which is usually the right move when f is a black box, noisy, or has no derivative.

Gradient Ascent and Descent

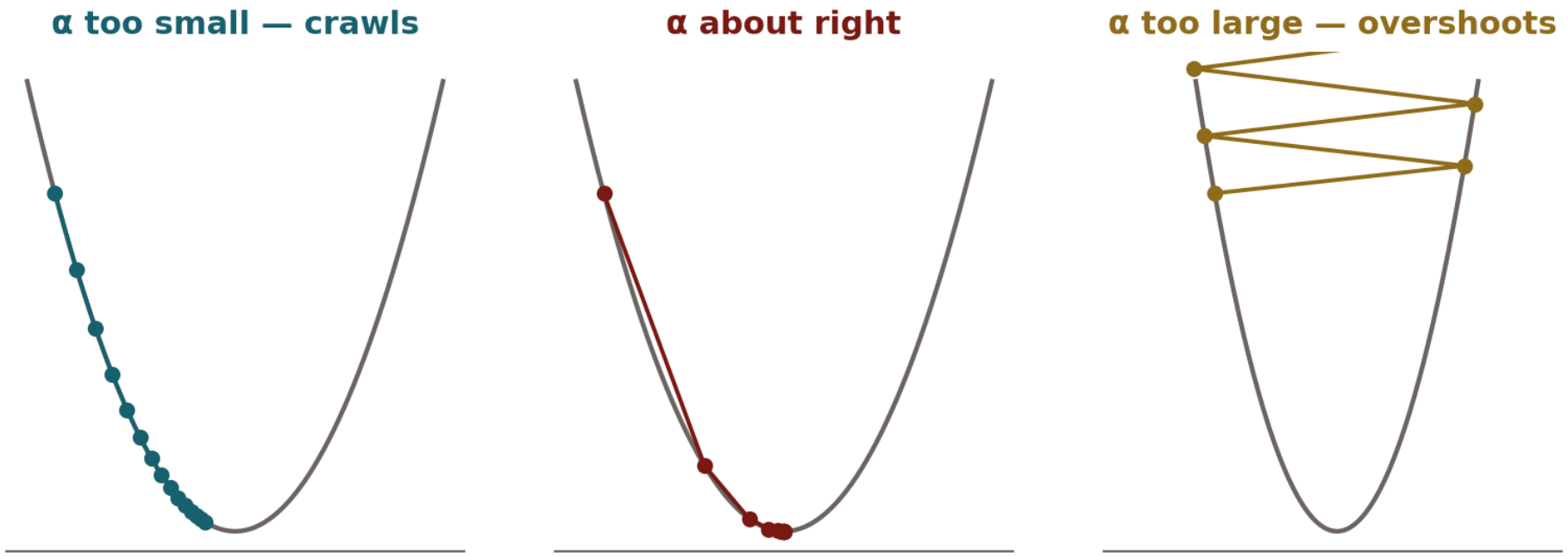
- $\nabla f(x) = (\partial f/\partial x_1, \partial f/\partial y_1, \partial f/\partial x_2, \ldots)$: the direction of steepest increase
- For the airports: $\partial f/\partial x_1 = \Sigma c \in C_1\ 2(x_1 - xc)$
- Set it to zero: $x_1 = (\Sigma c \in C_1\ xc)/|C_1|$
  - put each airport at the centroid of its own cities
  - but C_1 depends on where the airports are
  - so alternate: assign cities, re-centre, repeat: this is k-means, and it finds a local optimum only
- **When there is no closed form, iterate: $x \leftarrow x + \alpha\nabla f(x)$**



Same landscape traps as before — the start point decides the peak.

Getting the Step Size Right

- α too small: correct but slow. α too large: overshoot, oscillate, diverge
- Line search: keep doubling α while f improves, then back off: no hand-tuning needed
- Newton–Raphson uses curvature, $x \leftarrow x - H^{-1}(x)\nabla f(x)$, with $H_{ij} = \partial^2 f / \partial x_i \partial x_j$: far fewer steps, but H is n × n (hence BFGS and L-BFGS)



Local maxima, plateaux and ridges all survive the move to continuous space: restarts and annealing still earn their keep.

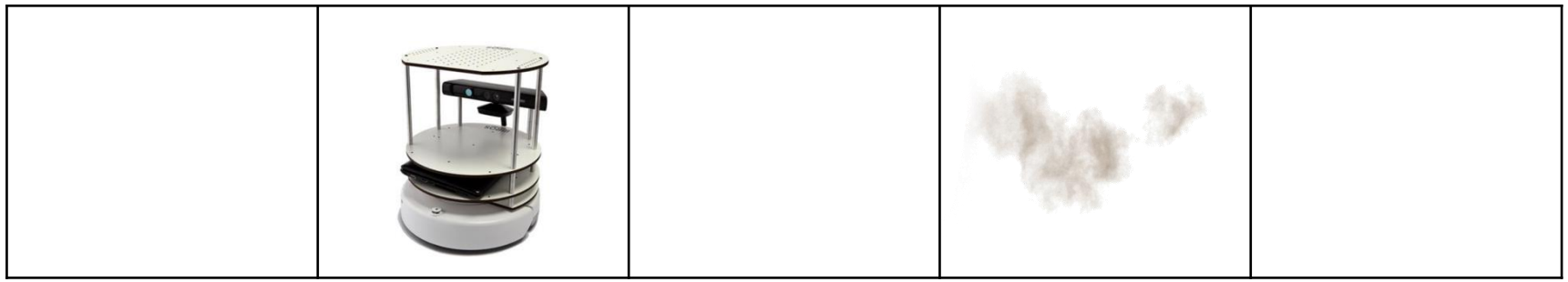
Constrained Optimization

- Constrained problems require solutions to satisfy hard constraints on the variables: the airports must be on land, the budget must balance
- Linear programming: linear constraints bounding a convex region, plus a linear objective
  - solvable in polynomial time in the number of variables
  - the best-studied class of optimization problems there is
- Convex optimization: a convex feasible region and a convex objective
  - strictly more general than linear programming, still polynomial
  - and the key property: every local optimum is a global optimum
- So where these forms fit, local search is not needed at all. Hill climbing, annealing and evolution are for everything else

Generalizations of the Search Problem

Problem Types and Solution Representations

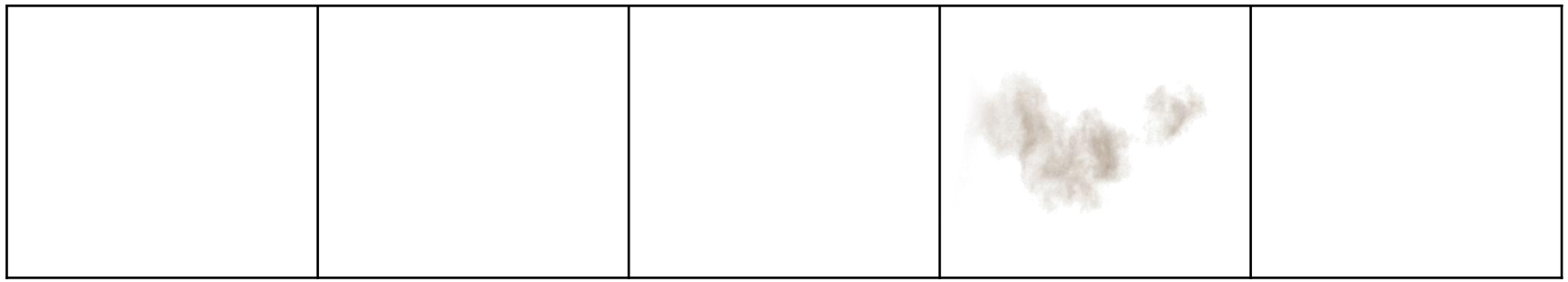
- Deterministic, fully observable problem
  - $T: S \times A \rightarrow S$; agent observes full state
  - "Classical planning": solution is a plan, or a sequence of actions



Solution plan: right, right, suck

Forms of Solutions

- Deterministic, fully observable
  - "Classical planning": solution is a plan, or a sequence of actions
- Deterministic, non-observable
  - "Conformant planning": solution is a plan, but often no solutions exist



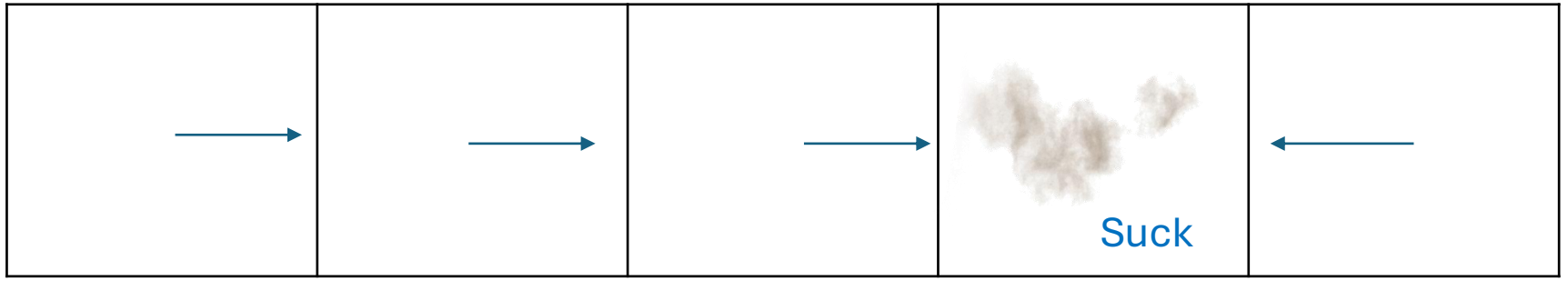
???

Right, Right, Right, Right, Suck, Left, Suck, Left, Suck, Left, Suck, Left, Suck

Forms of Solutions

- Deterministic, fully observable
  - "Classical planning": solution is a plan, or a sequence of actions
- Deterministic, non-observable
  - "Conformant planning": solution is a plan, but often no solutions exist
- Non-deterministic, fully observable:
  - "Contingent planning": solution can be a policy (a function $S \rightarrow A$)

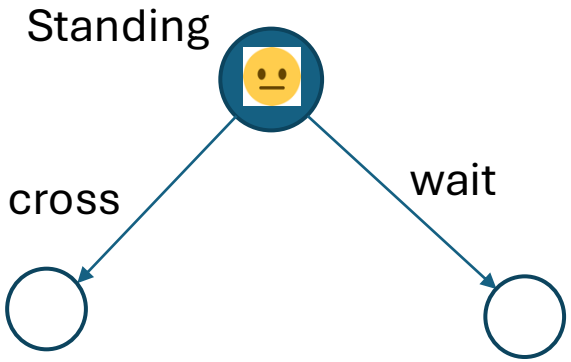
Non-determinism: a move action may fail



Dealing with Non-Determinism

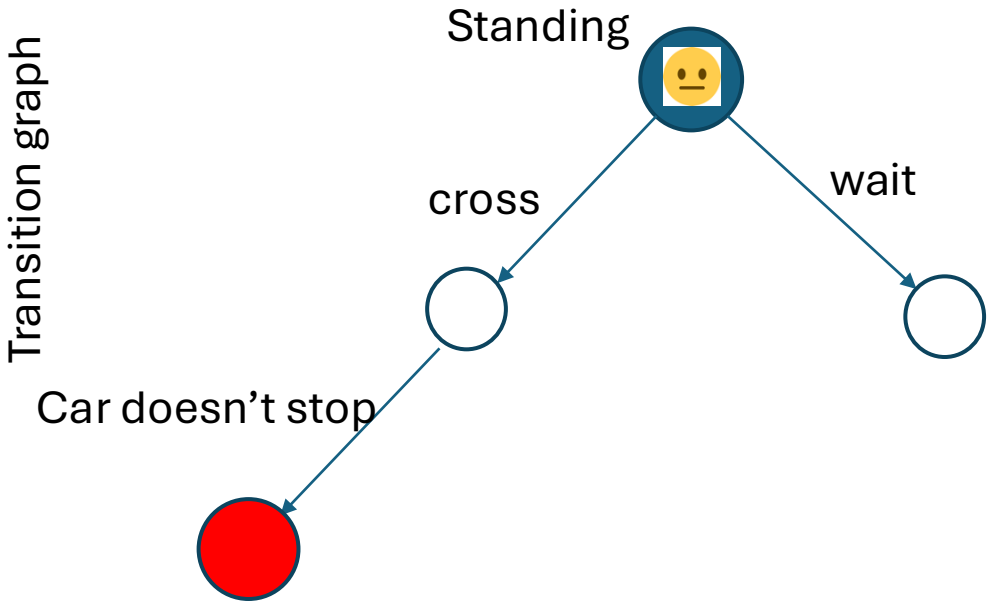
- Actions: To cross or not to cross
- There's a chance of being run over

Transition graph



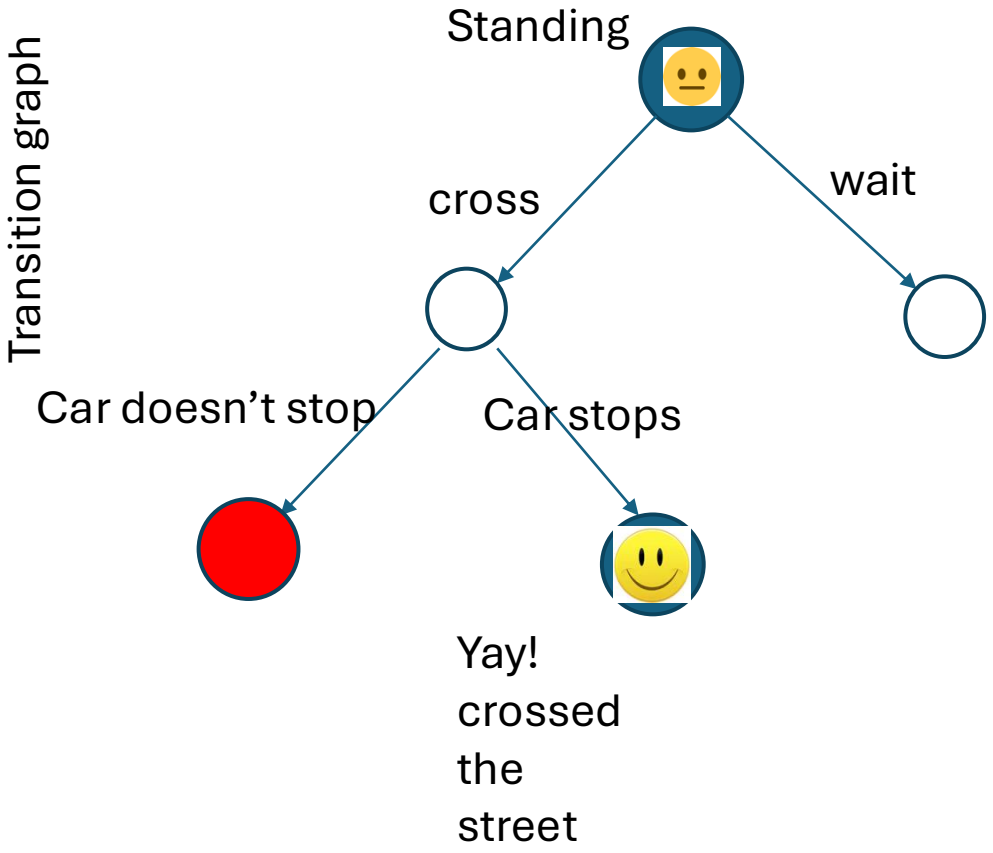
Dealing with Non-Determinism

- Actions: To cross or not to cross
- There's a chance of being run over



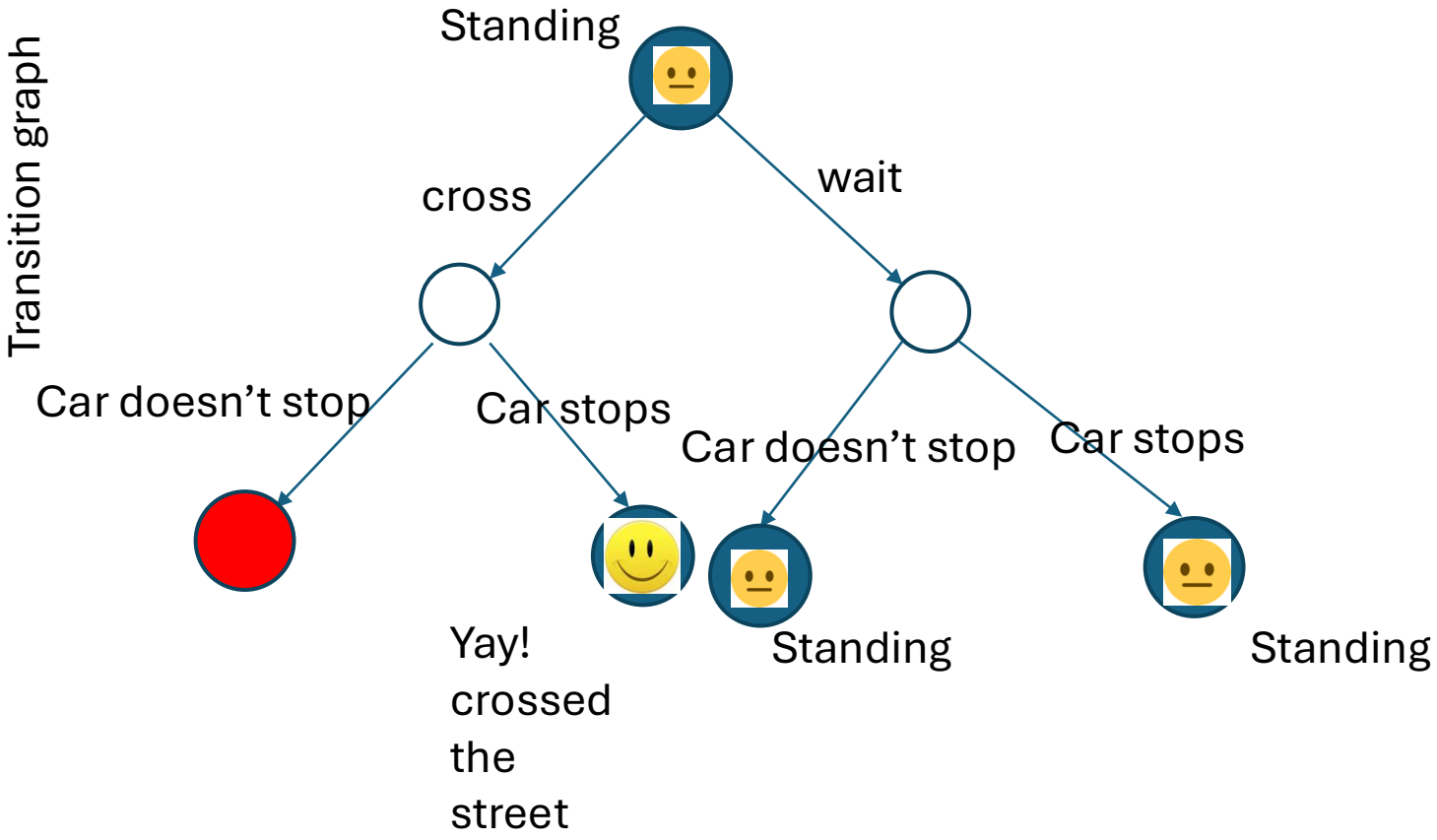
Dealing with Non-Determinism

- Actions: To cross or not to cross
- There's a chance of being run over



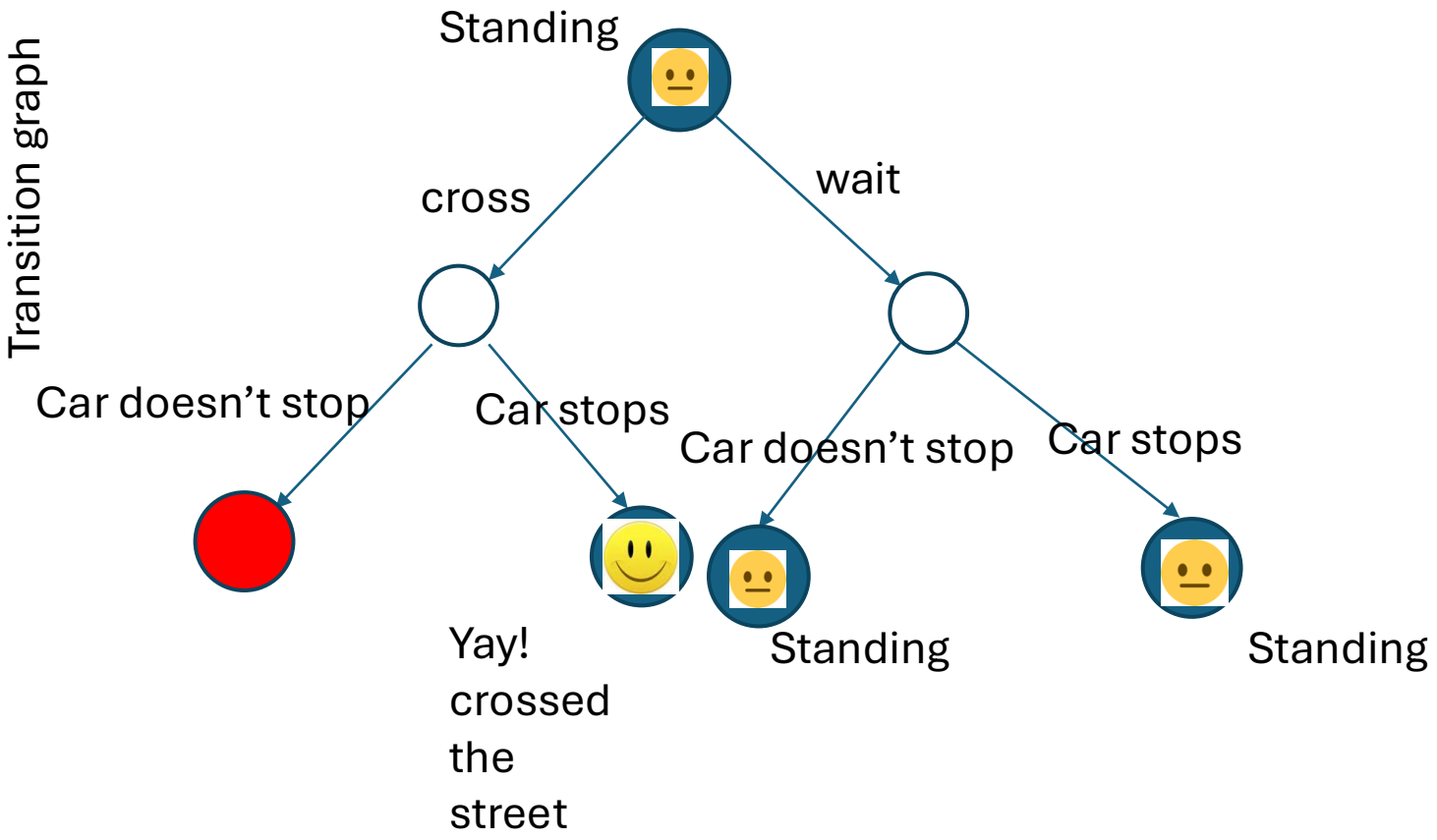
Dealing with Non-Determinism

- Actions: To cross or not to cross
- There's a chance of being run over



Dealing with Non-Determinism

- Actions: To cross or not to cross
- There's a chance of being run over

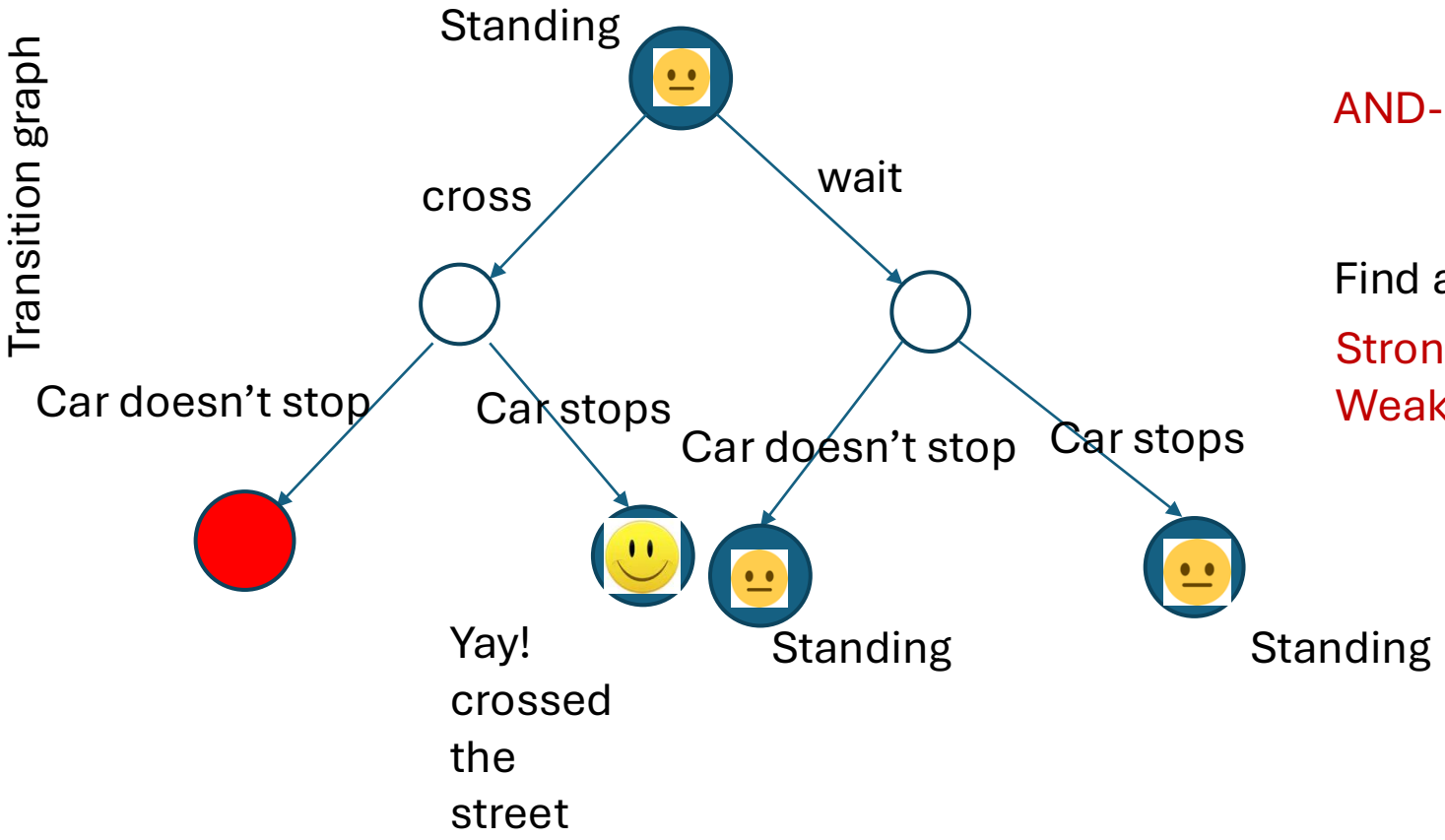


Linear plans are no longer sufficient

Plans need to consider all contingencies, indicate correct behavior for each

Dealing with Non-Determinism

- Actions: To cross or not to cross
- There's a chance of being run over



Changes from previous graphs:

- OR nodes
- AND nodes

AND-OR path: must include any successor of OR node
must include all successors of AND nodes

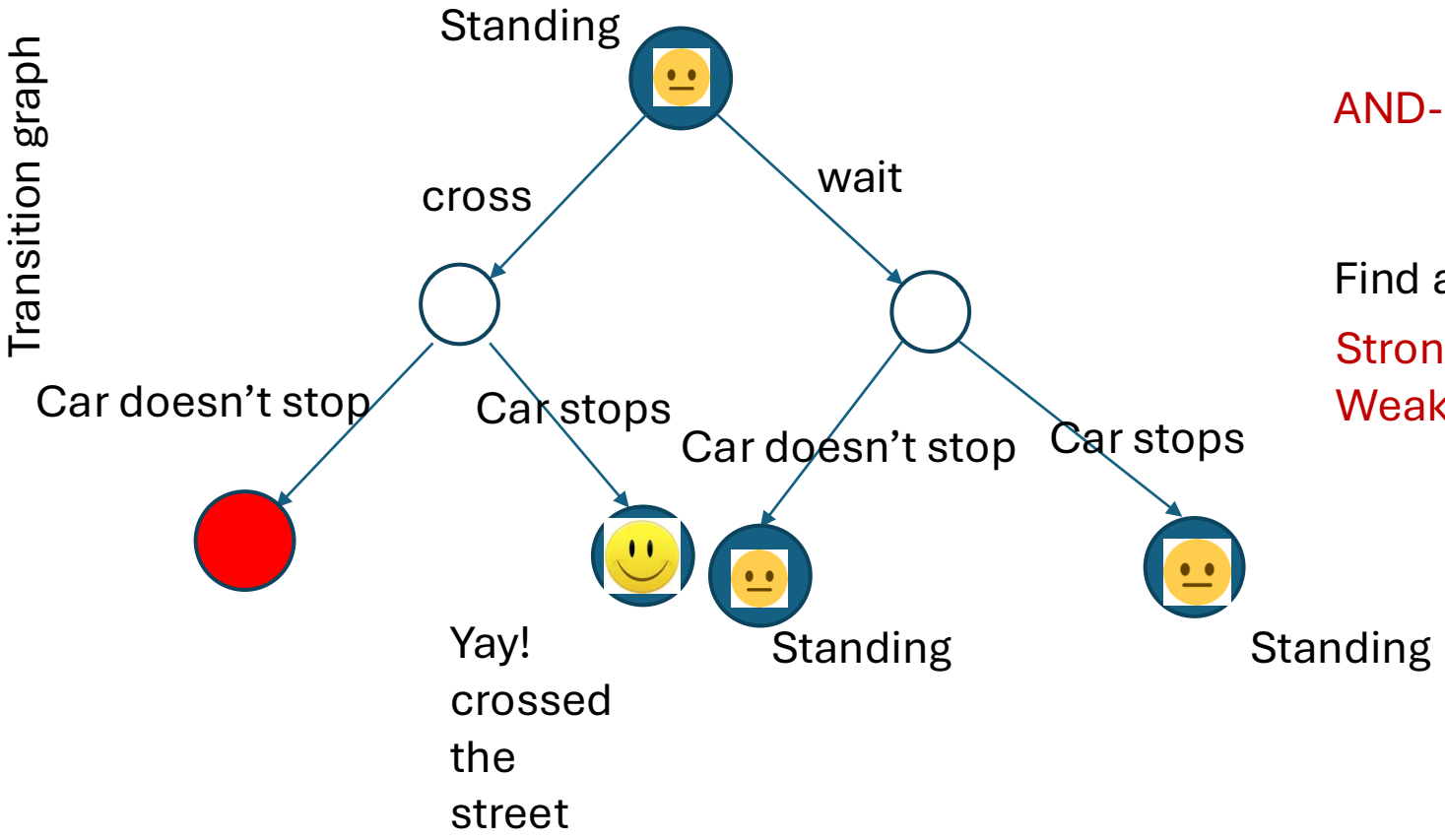
Find an AND-OR path that ends at the goal

Strong plan: no cycles, all leaves are goals

Weak plan: no cycles, some leaves are goals

Dealing with Non-Determinism

- Actions: To cross or not to cross
- There's a chance of being run over



Changes from previous graphs:

- OR nodes
- AND nodes

AND-OR path: must include any successor of OR node
must include all successors of AND nodes

Find an AND-OR path that ends at the goal

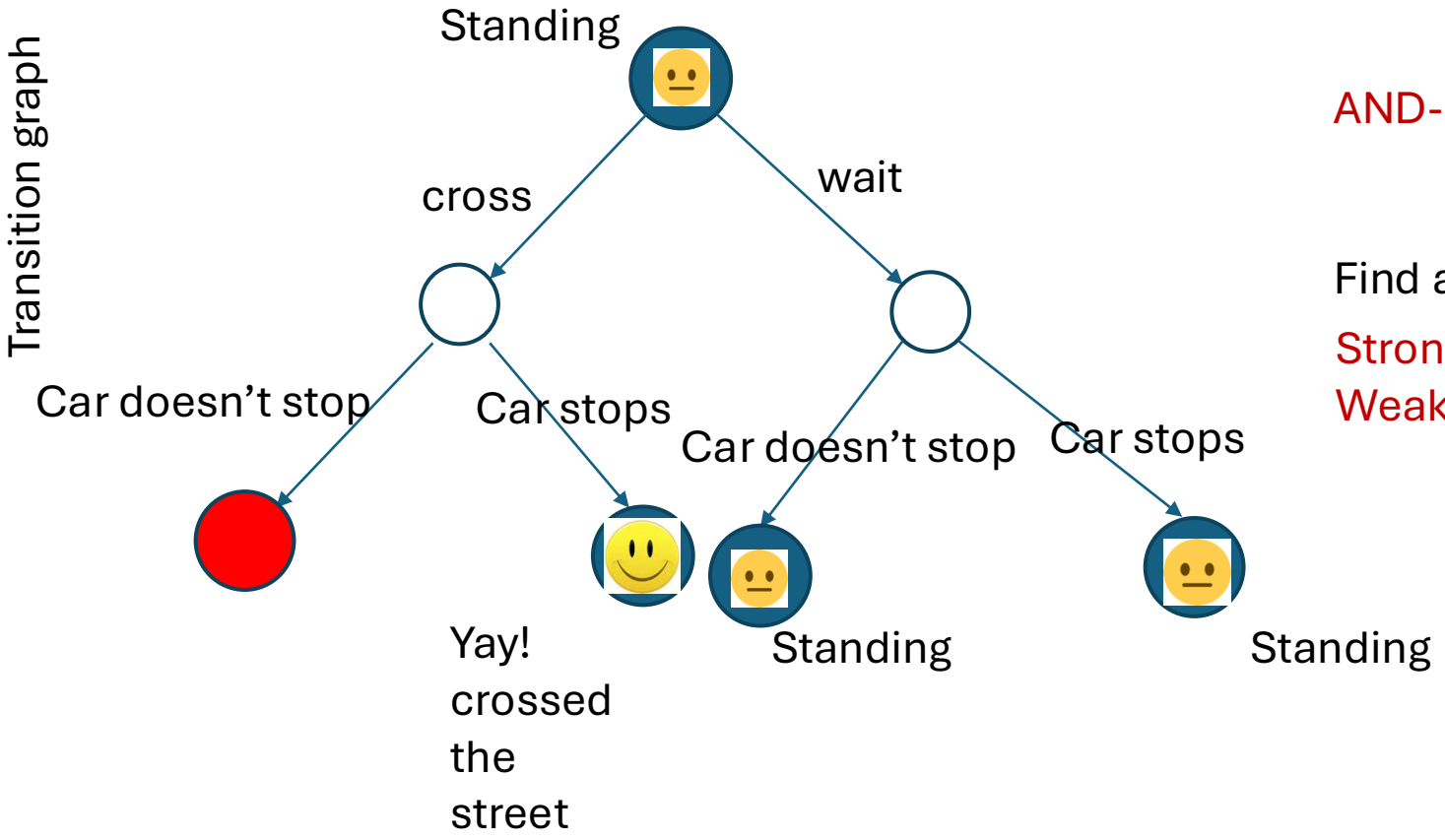
Strong plan: no cycles, all leaves are goals

Weak plan: no cycles, some leaves are goals

Are there any strong plans here?
Any weak plans?

Dealing with Non-Determinism

- Actions: To cross or not to cross
- There's a chance of being run over



Changes from previous graphs:

- OR nodes
- AND nodes

AND-OR path: must include any successor of OR node
must include all successors of AND nodes

Find an AND-OR path that ends at the goal

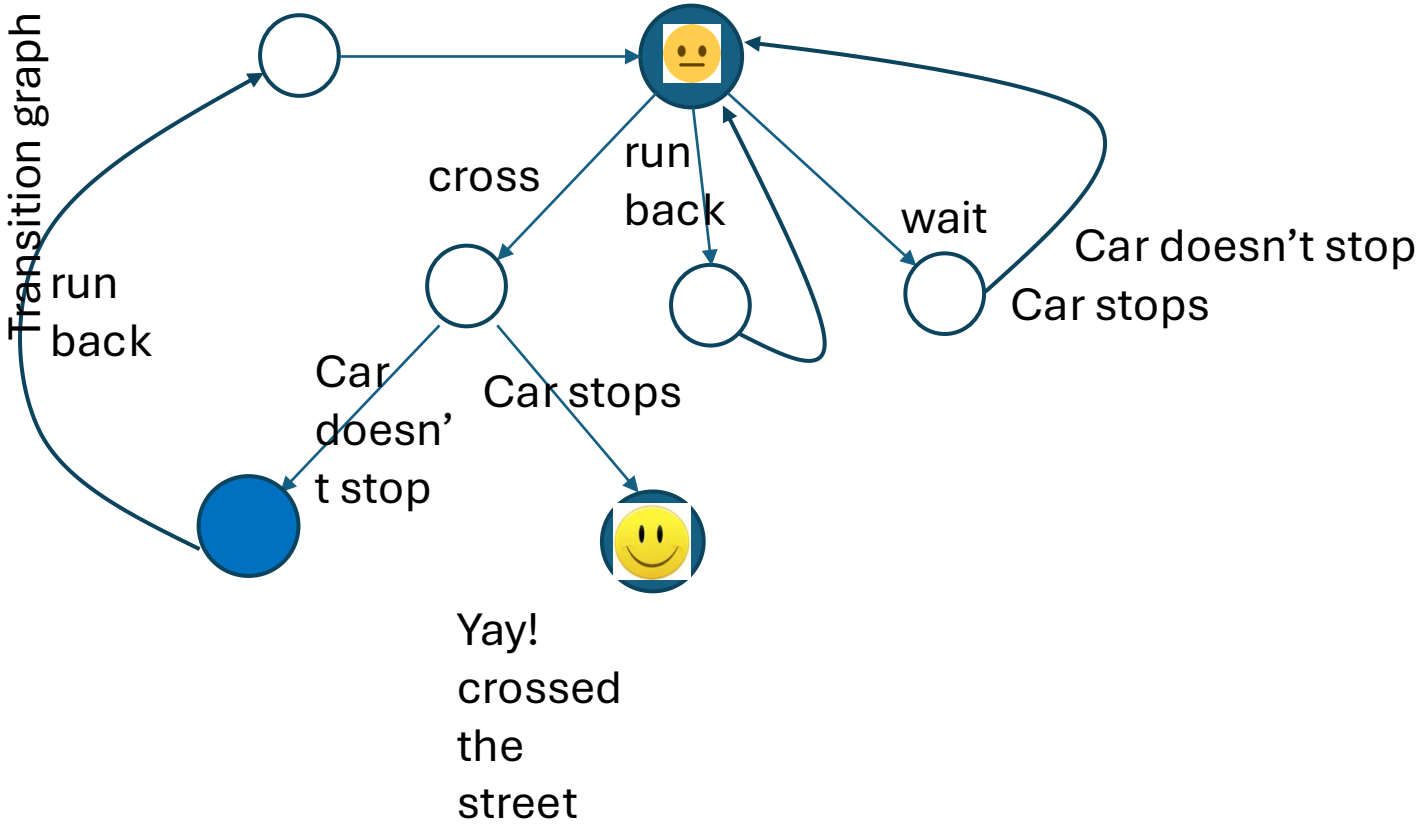
Strong plan: no cycles, all leaves are goals
Weak plan: no cycles, some leaves are goals

Are there any strong plans here?
Any weak plans?

What if we allow cycles?

Dealing with Non-Determinism

- Actions: cross, wait, run back
- There's a chance of being run over



Strong cyclic plan:

AND-OR path, may include cycles

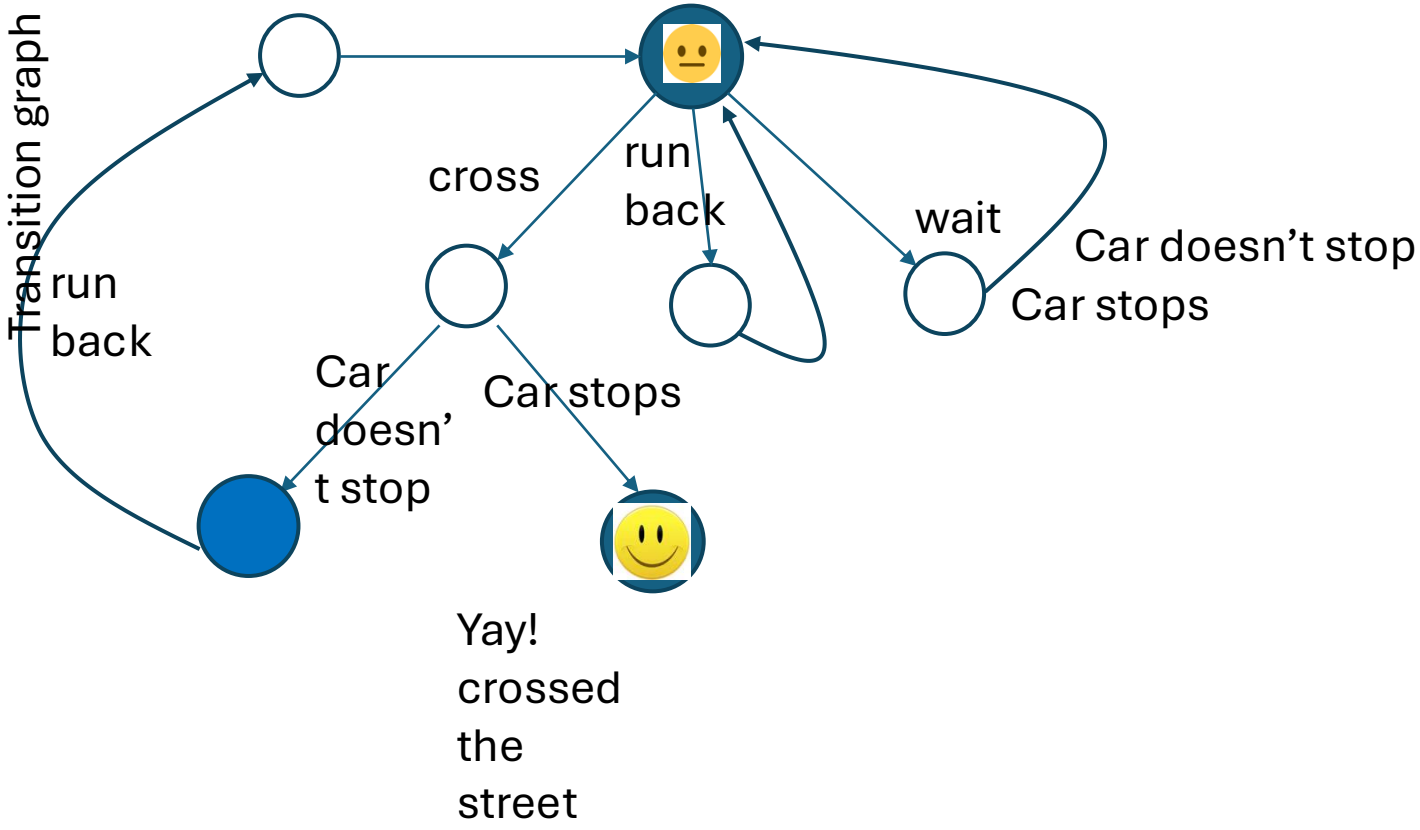
All leaf nodes must be goal nodes

Strong cyclic plans are often possible

Dealing with Non-Determinism

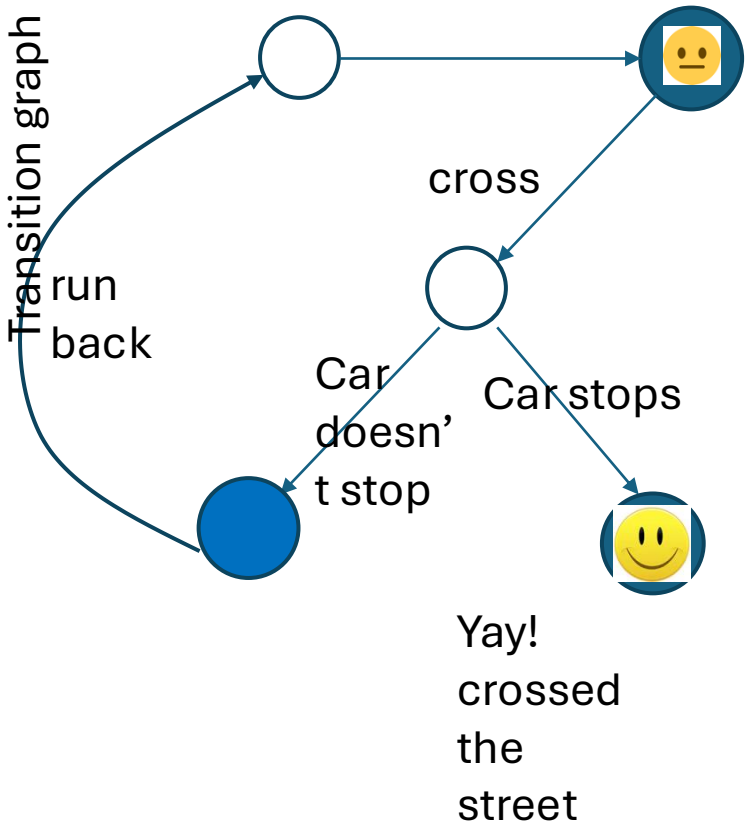
- Actions: cross, wait, run back
- There's a chance of being run over

New action: run back
Can be executed in any state



Dealing with Non-Determinism

- Actions: cross, wait, run back
- There's a chance of being run over



Strong cyclic plan:

AND-OR path, may include cycles

All leaf nodes must be goal nodes

Strong cyclic plans are often possible

But they offer very weak guarantees:

at any time during execution, agent has the possibility of eventually reaching the goal

No bounds ☹

Limitations of Strong Cyclic Plans

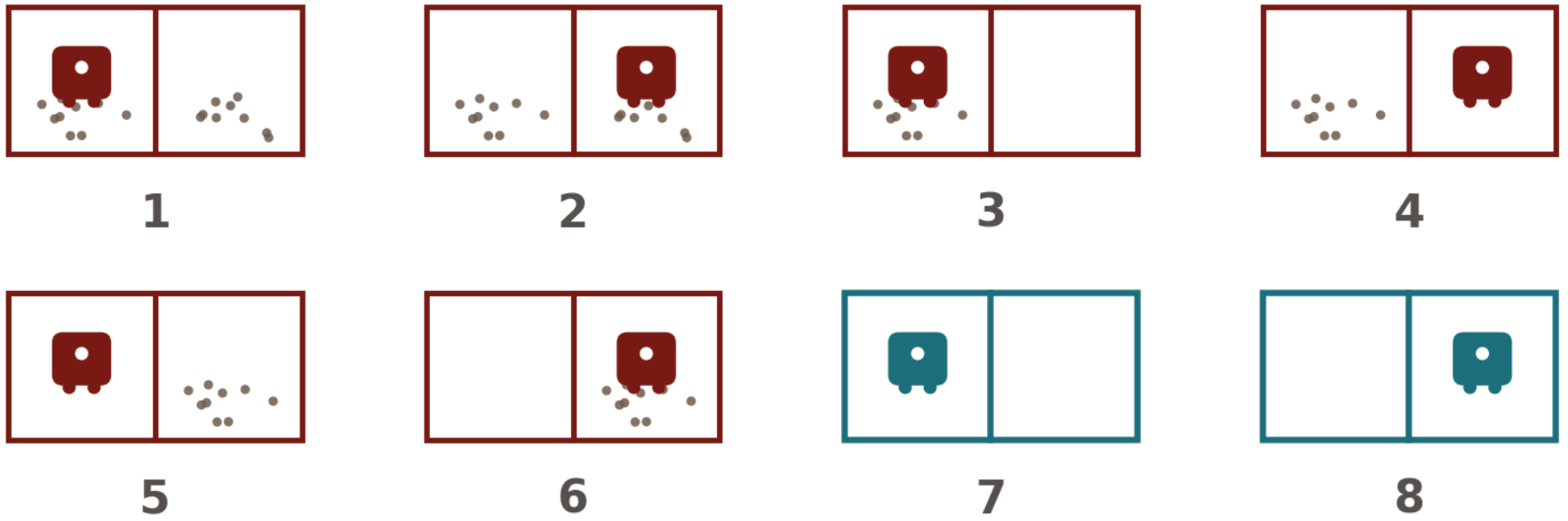
- Hard to evaluate which solution is better
- Poor guarantee: unbounded wait!

Another Example: The Erratic Vacuum World

When one action can turn out several ways

Eight States, Three Actions

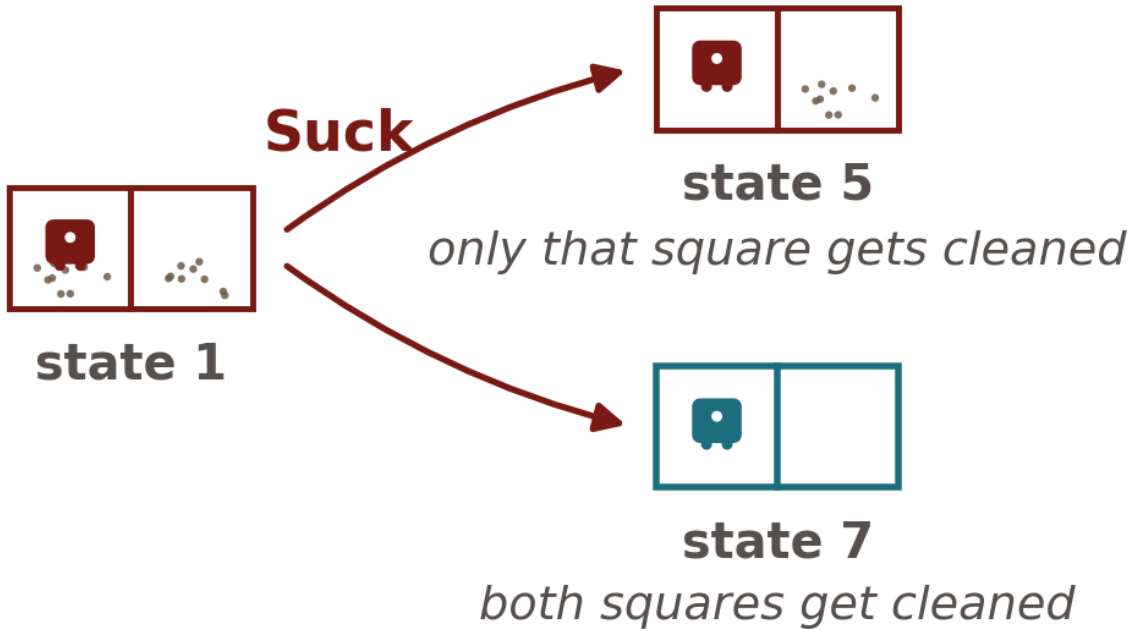
- Two squares, each clean or dirty, and the agent standing in one of them
- Actions are Left, Right and Suck, and the goal is both squares clean



states 7 and 8 are the goal states

Suck Does Not Always Do the Same Thing

- On a dirty square, Suck cleans it and sometimes cleans the other square too
- On a clean square, Suck sometimes deposits dirt instead
- So RESULTS(1, Suck) = {5, 7}, a set of states rather than a single state



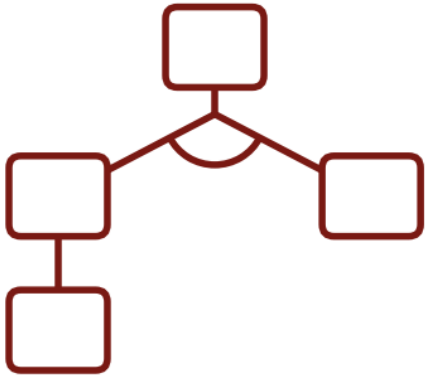
A Plan Is No Longer a Straight Line

- A fixed sequence can fail, since the agent cannot know in advance which outcome it will get
- The agent perceives its state after acting, so the plan can branch on what it sees
  - [Suck, if State = 5 then [Right, Suck] else [ ]]

deterministic world

a plan is a sequence

nondeterministic world

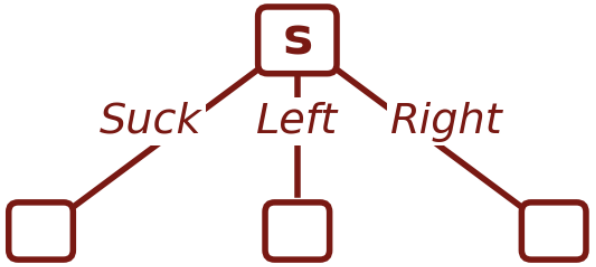


a plan is a tree

AND-OR Search Trees

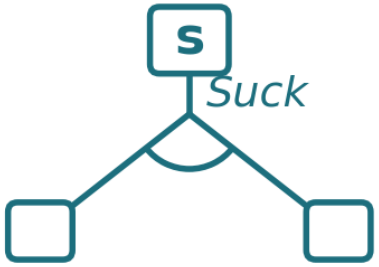
- OR nodes are the agent's own choice, so one branch is enough
- AND nodes are the outcomes the environment may pick, so every branch has to be covered
- The two kinds of node alternate all the way down the tree

OR node



the agent chooses,
so one branch is enough

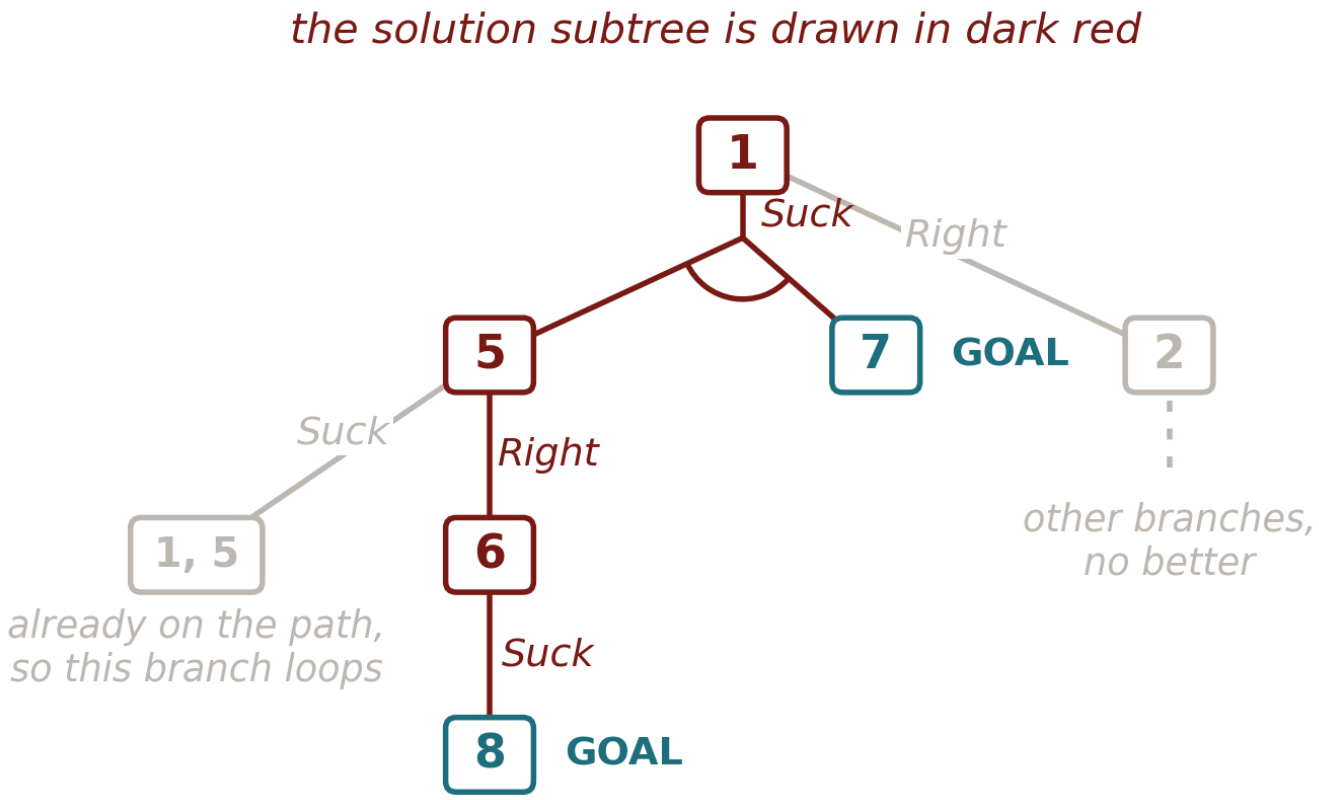
AND node



the environment chooses,
so cover every branch

What Counts as a Solution

- A solution is a subtree of the search tree, not a path through it
  - One action chosen at every OR node it contains
  - Every outcome included at every AND node
  - A goal state at every leaf
- Reading the marked subtree off gives back the branching plan



Searching an AND-OR Tree

- Depth first, with the two node types calling each other
- A state already on the current path means a loop, so that branch fails
- An OR node needs one branch to work, an AND node needs all of them

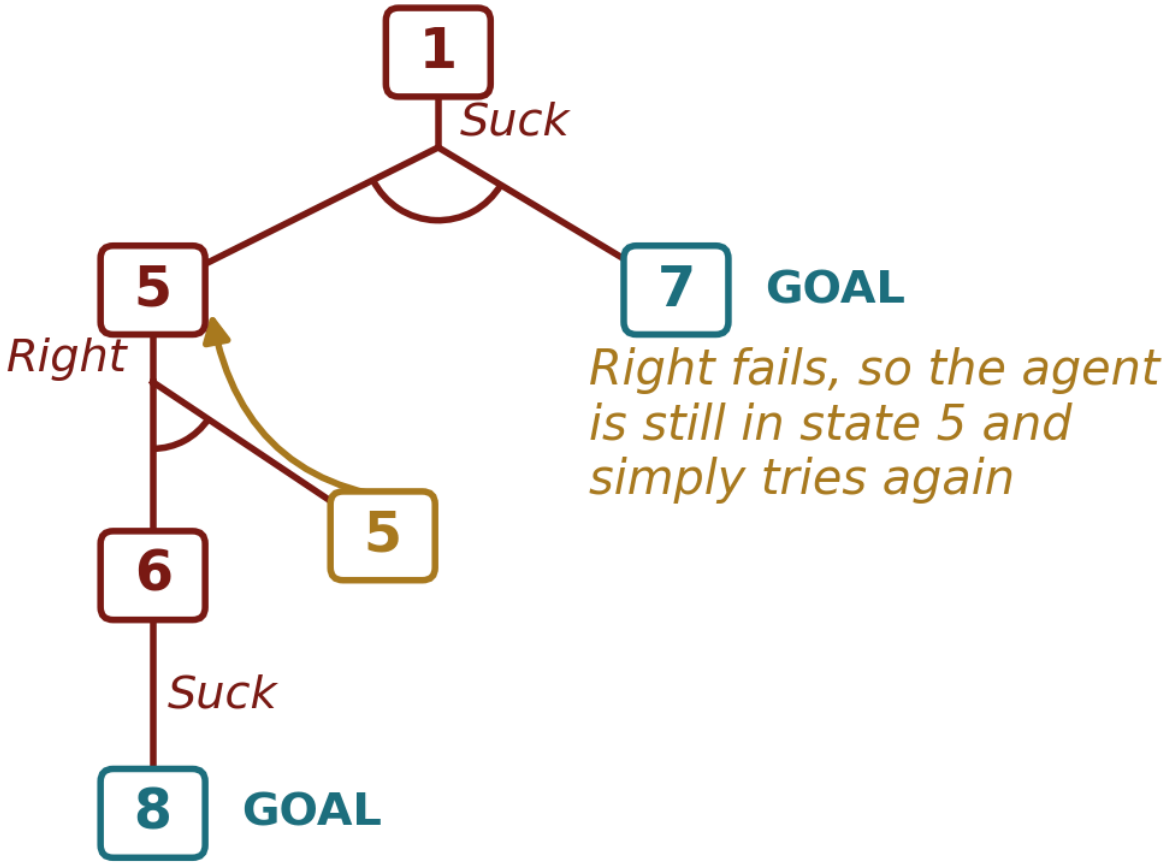
```
OR-SEARCH(state, problem, path)
  if problem.IS-GOAL(state) then return the empty plan
  if state is on path then return failure
  for each action in problem.ACTIONS(state) do
    plan ← AND-SEARCH(RESULTS(state, action), problem,
       [state] + path)
    if plan is not failure then return [action] + plan
  return failure

AND-SEARCH(states, problem, path)
  for each s in states do
    plan(s) ← OR-SEARCH(s, problem, path)
    if plan(s) is failure then return failure
  return the branching plan built from all the plan(s)
```

Try, Try Again

When the only way through is to repeat yourself

The Slippery Vacuum World

- Movement is unreliable now, so Right sometimes leaves the agent where it was
- RESULTS(5, Right) = {5, 6}
- No acyclic solution exists, since every finite plan runs out of moves on the branch where the wheels keep slipping



Cyclic Plans

- Let a plan point back into itself, so a failed attempt is simply repeated

```
[Suck, while State = 5 do Right, Suck]
```

- It is a solution when every leaf is a goal and every leaf is reachable from every point in the plan
- The guarantee holds only if the failure does not repeat forever, so the action must eventually work
- This is the strong cyclic plan we met earlier, now with the condition stated precisely