EC207a
AI and its Application

Dr. Sachin Kumar Jain

PDPM IIITDM Jabalpur

skjain@iiitdmj.ac.in

Teachers open the door but you must enter by yourself
---Chinese Proverb

Agent types

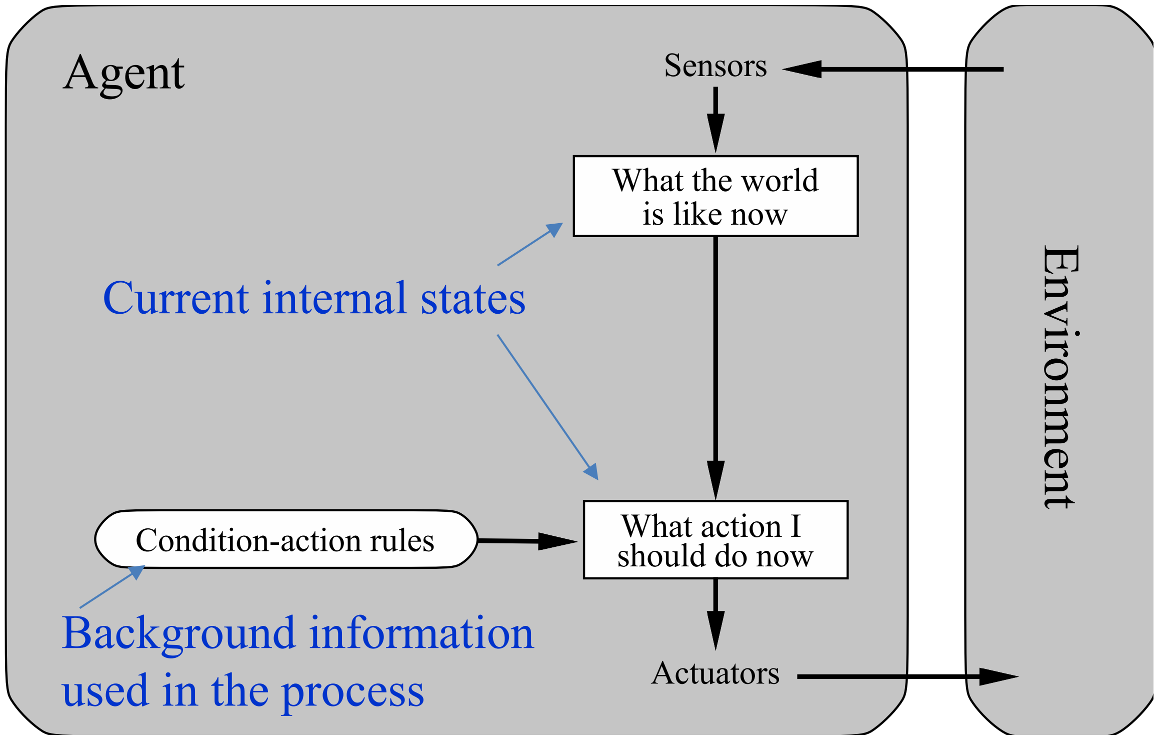
- In order of increasing generality and complexity, the agent programs are classified as:
  - Simple reflex agents
  - Reflex agents with state
  - Goal-based agents
  - Utility-based agents

Simple reflex agents

These agents select action on the basis of the current percept, ignoring the rest of the percept history. For example: vacuum agent that just checks the current location and dirt in it.

It follows ***condition-action rule***, in which a condition triggers an established connection in the agent program to the action.

Humans also have many such connections, some of which are learned responses (***if*** car-in-front-is-braking ***then*** initiate-braking) and some of which are innate reflex (blinking eye on sudden …).

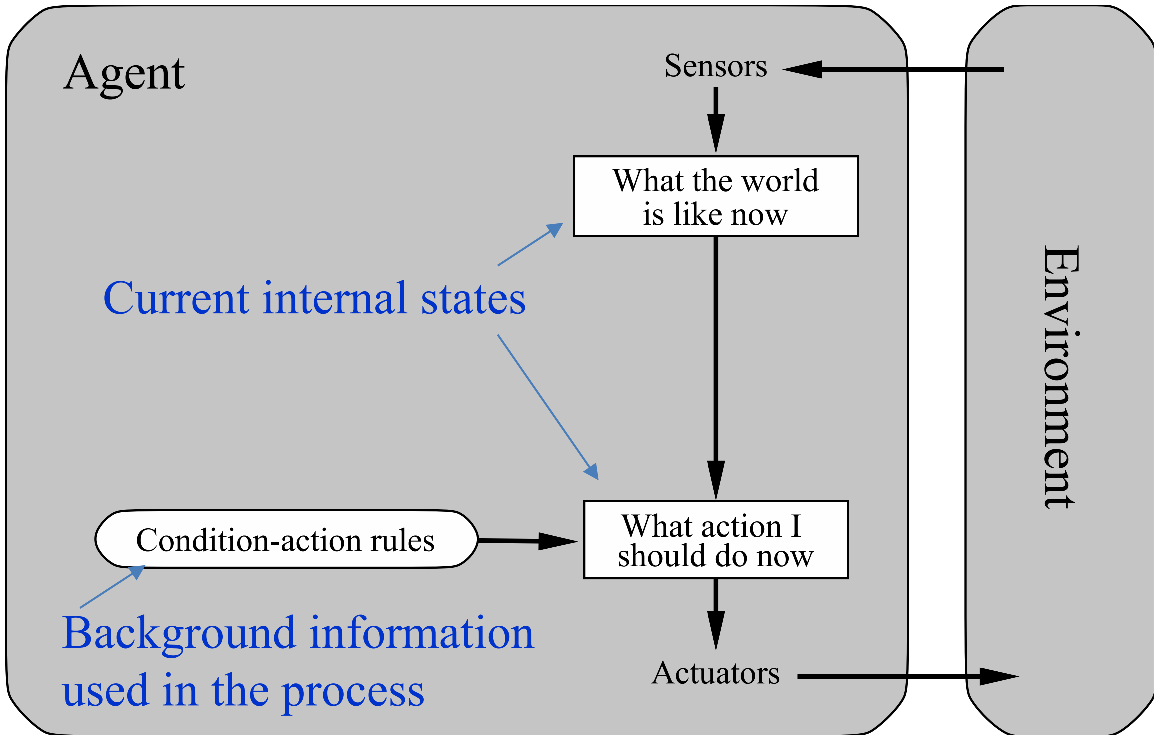

Agent
Sensors
What the world is like now
Current internal states
Condition-action rules
What action I should do now
Background information used in the process
Actuators
Environment

Simple reflex agents

The advantage of Simple reflex agents: Simple. But they turn out to be of very limited intelligence.

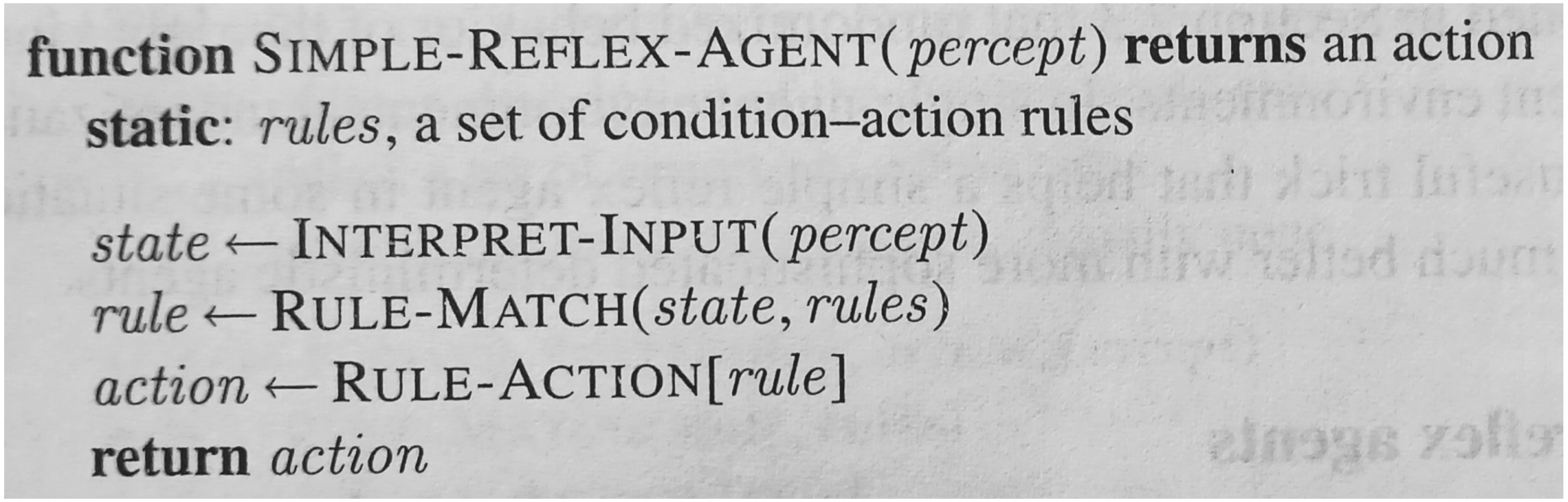
A little bit of unobservability can cause serious trouble.

For example: the vacuum agent has only dirt sensor working, then two possible percept will be: *Dirty* and *Clean*. It will *Suck* in response to *Dirty*, but it will fail to move *left* or *right*, in response to *Clean*.

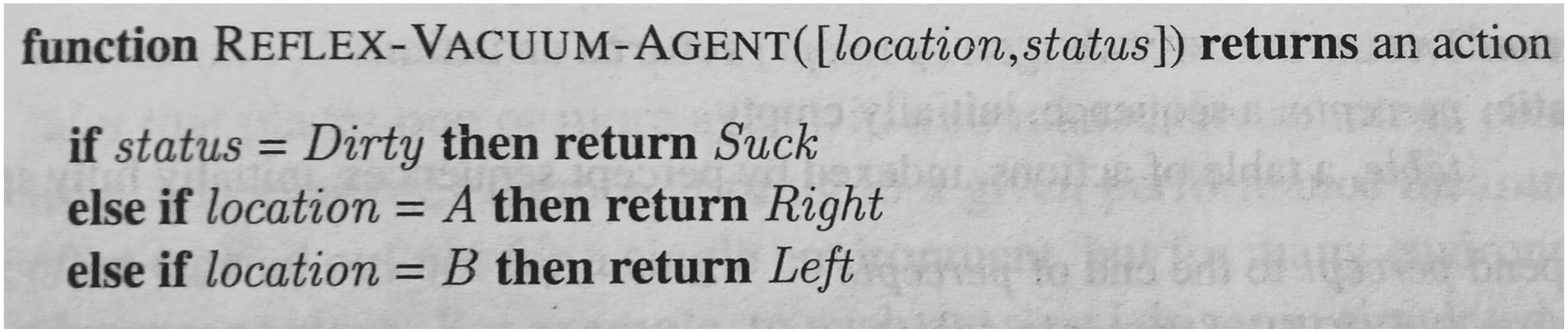


Agent
Sensors
What the world is like now
Current internal states
Condition-action rules
What action I should do now
Background information used in the process
Actuators
Environment

Simple reflex agents



```
function SIMPLE-REFLEX-AGENT(percept) returns an action
  static: rules, a set of condition–action rules

  state ← INTERPRET-INPUT(percept)
  rule ← RULE-MATCH(state, rules)
  action ← RULE-ACTION[rule]
  return action
```



```
function REFLEX-VACUUM-AGENT([location,status]) returns an action

  if status = Dirty then return Suck
  else if location = A then return Right
  else if location = B then return Left
```

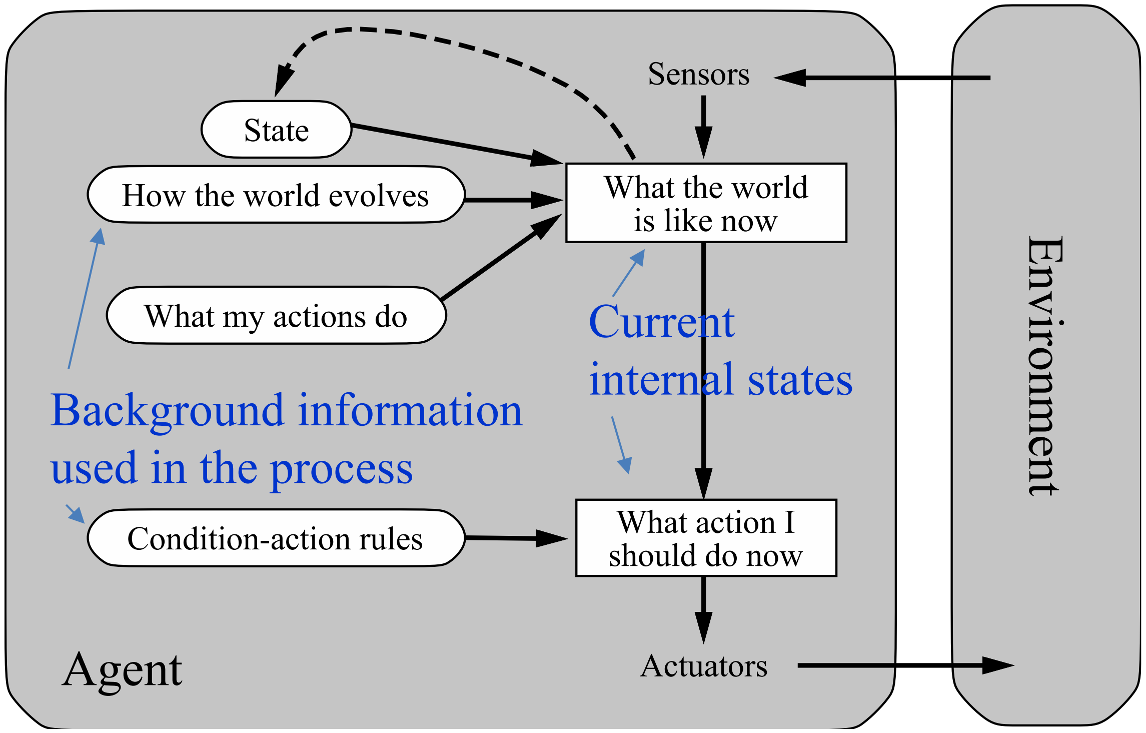
Reflex agents with state

These are known as **model-based** reflex agents. It maintains internal state that depend on the percept history, and thereby reflects at least some of the unobserved aspects of the current state.

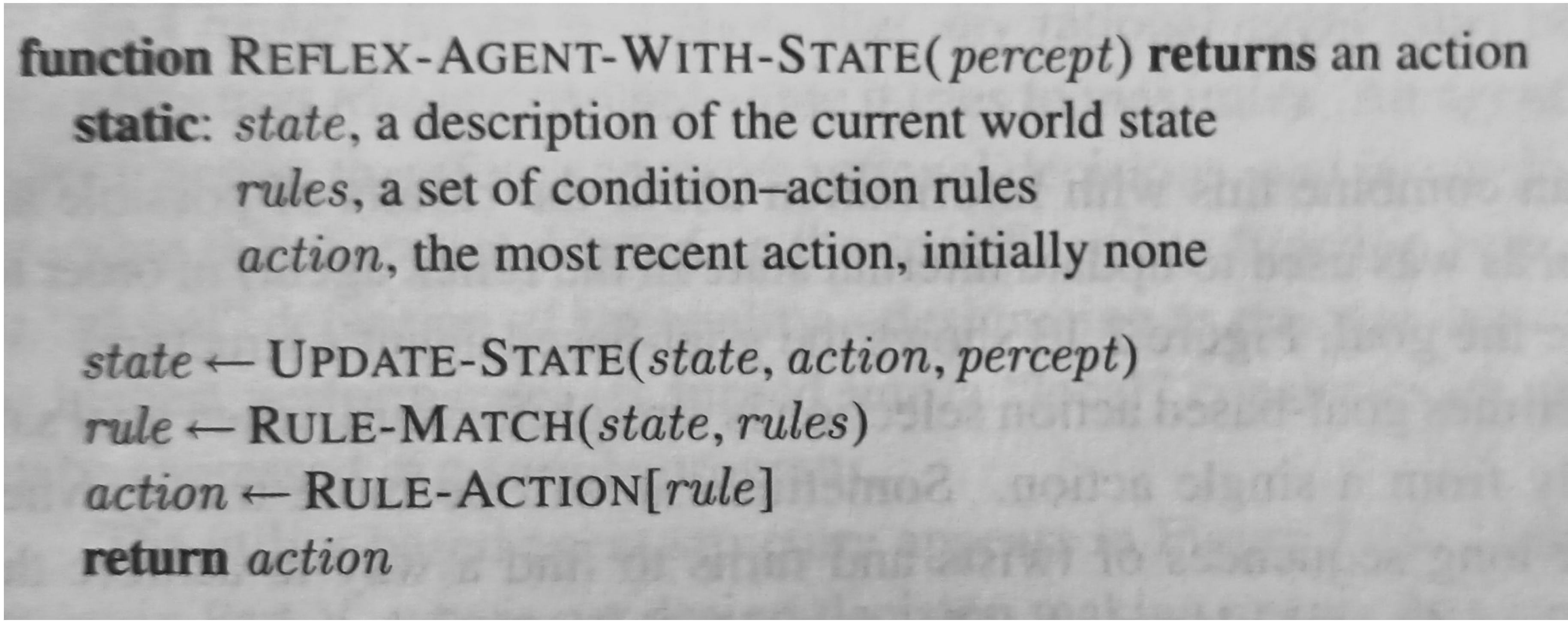
Updating this internal state information requires two kinds of knowledge to be encoded in the agent program.

1. Information about how world evolves.
2. Information about how agent's own action affect the world.

This is implemented using model of the world.



Reflex agents with state



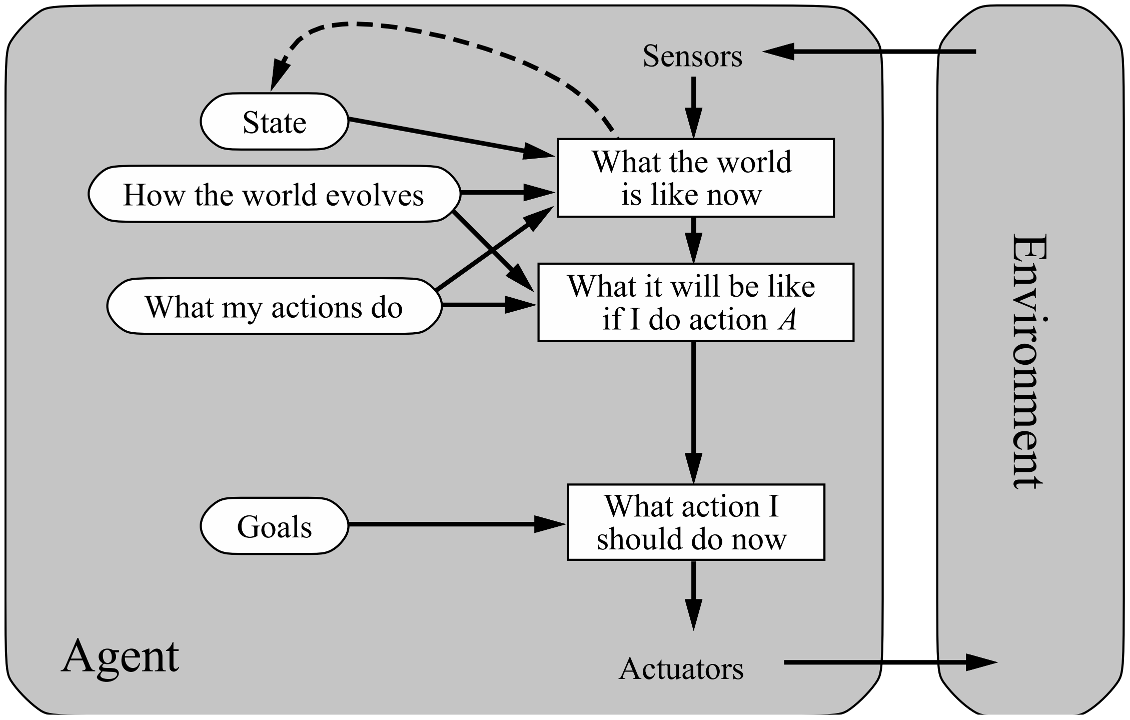
```
function REFLEX-AGENT-WITH-STATE(percept) returns an action
    static: state, a description of the current world state
            rules, a set of condition–action rules
            action, the most recent action, initially none

    state ← UPDATE-STATE(state, action, percept)
    rule ← RULE-MATCH(state, rules)
    action ← RULE-ACTION[rule]
    return action
```

Goal-based agents

The state of the environment is not always enough to decide what to do. The agent needs some sort of goal information that describes situations that are desirable.

The agent program can combine this with information about the results of possible actions in order to choose actions that achieve the goal.

Decision making of this kind is fundamentally different from the ***condition-action rule***, as it considers the future:- "***what will happen if I do such and such***", and "***will that make me happy***"?


Sensors
State
What the world is like now
How the world evolves
What it will be like if I do action A
What my actions do
Goals
What action I should do now
Agent
Actuators
Environment

Goal-based agents

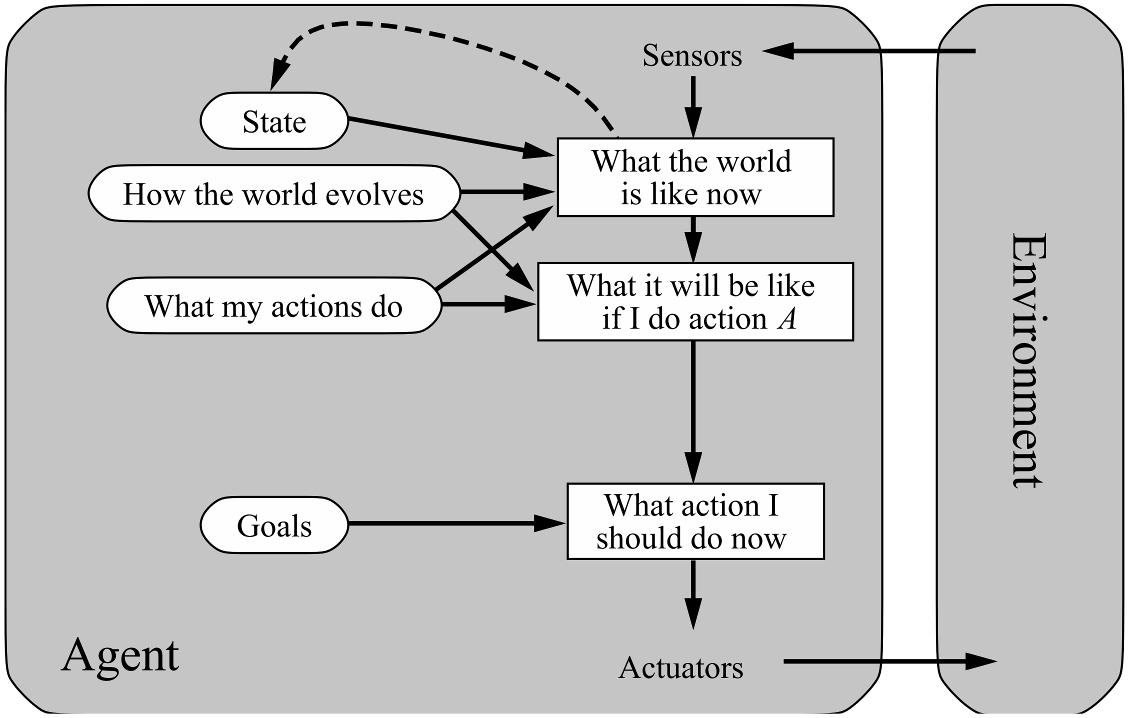
The state of the environment is not always enough to decide what to do. The agent needs some sort of goal information that describes situations that are desirable.

The agent program can combine this with information about the results of possible actions in order to choose actions that achieve the goal.

Pro: More flexible, use goal to index into actions that might achieve it, e.g.

"Have milk" → "buy milk"

Con: Cannot handle tradeoffs among goals, appears less efficient, failure probability etc.


Sensors
State
How the world evolves
What my actions do
What the world is like now
What it will be like if I do action A
Goals
What action I should do now
Agent
Actuators
Environment

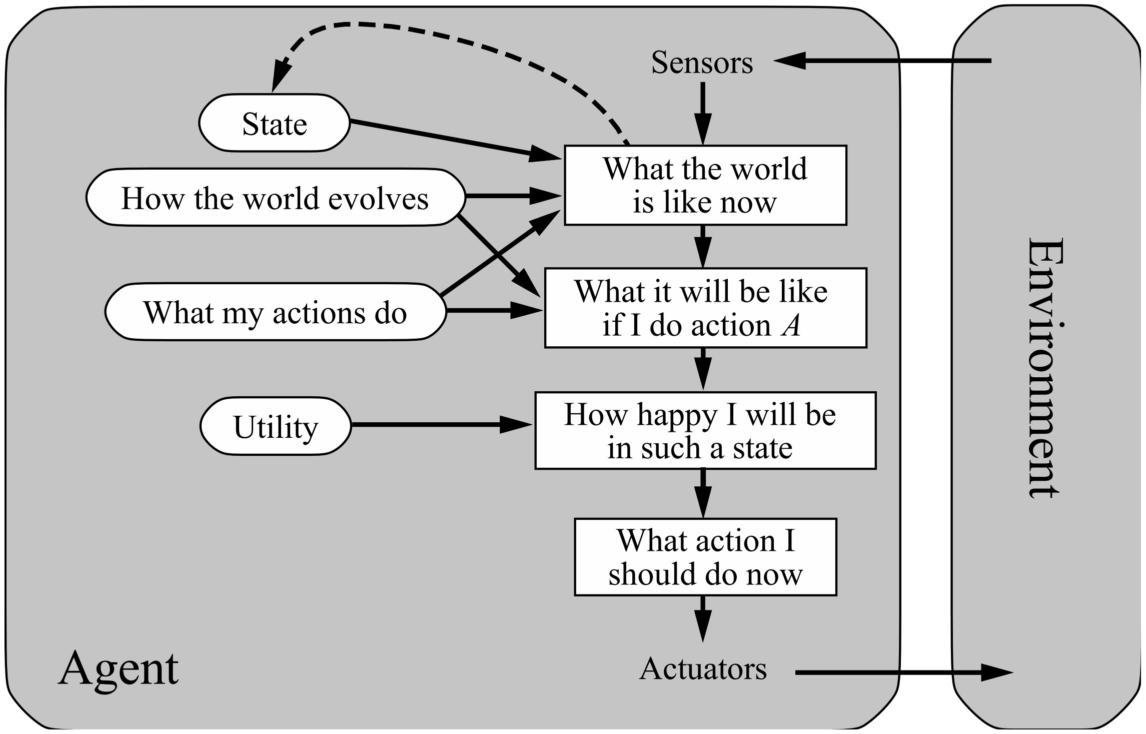
Utility-based agents

Goals alone are not really enough to generate high-quality behavior in most environments. A **utility function** maps a state onto a real number, which describes the associated degree of happiness. It allows handling conflicting goals (e.g. speed/time vs safety in drive). Utility is a numeric scale.

A complete specification of the utility function allows rational decisions with multiple and/or conflicting goals.

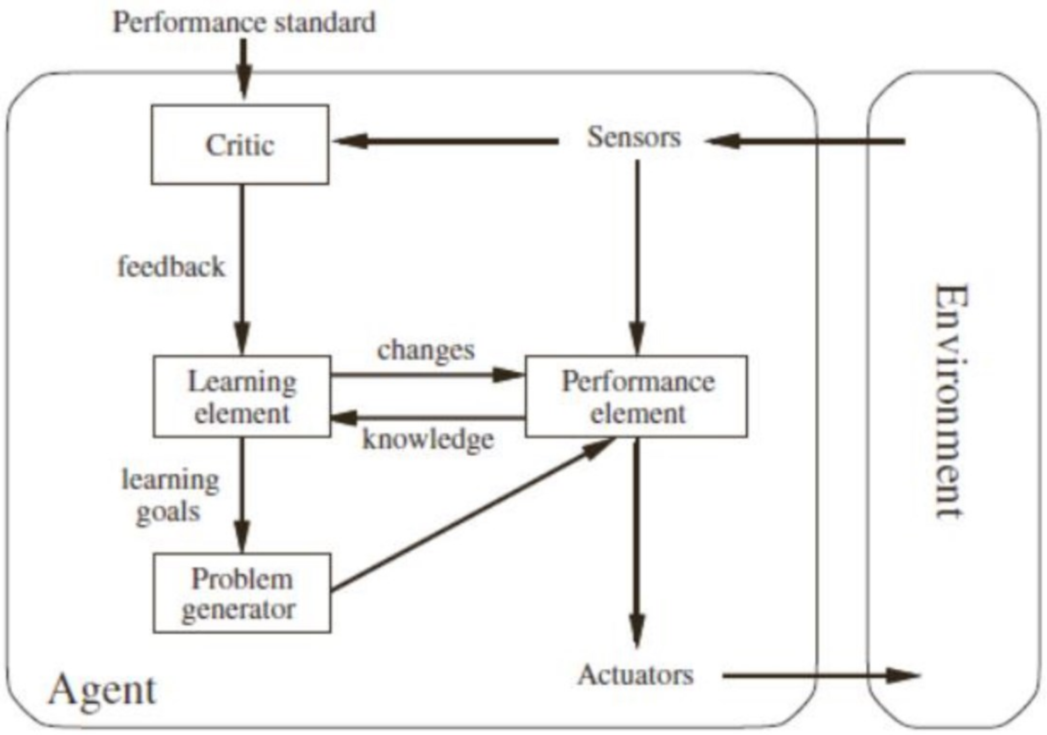
Pro: can compute expected values for actions, handle tradeoffs and uncertainty.

Con: cannot (easily) index into actions.


Sensors
State
What the world is like now
How the world evolves
What it will be like if I do action *A*
What my actions do
Utility
How happy I will be in such a state
What action I should do now
Agent
Actuators
Environment

Learning agents

- Learning element is responsible for making improvements.
- Performance element is responsible for selecting external actions.
- The learning element uses feedback from critic on how agent is doing and determines how performance element should be modified to do better in future.
- Problem generator is responsible for suggesting actions that will lead to new and informative experiences.



Summary

- An ***agent*** interacts with an ***environment*** through ***sensors*** and ***actuators***
- The ***agent function***, implemented by an ***agent program*** running on a ***machine***, describes what the agent does in all circumstances
- Rational agents choose actions that maximize their expected utility
- PEAS descriptions define task environments; precise PEAS specifications are essential and strongly influence agent designs
- More difficult environments require more complex agent designs and more sophisticated representations

Assignment #1

1. Develop a PEAS description of the task environment:

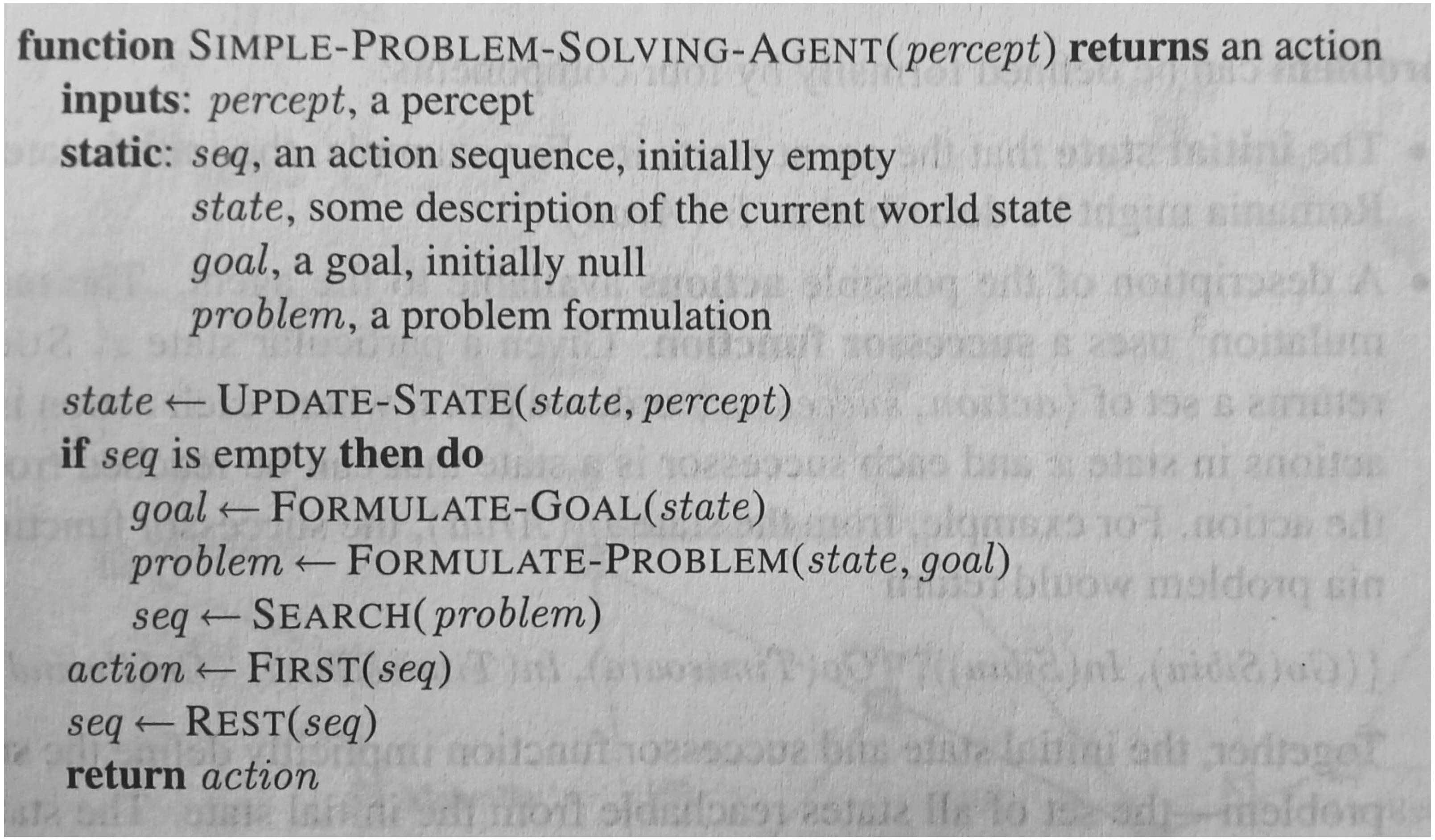
   A. English tutor

   B. Automatic vehicle

2. Design all four types of AI agents for an automatic car and compare their performance on different parameters (e.g. simplicity, robustness (performance when any sensor is having noise, etc.), goal, safety, comfort, etc.

Problem-solving agents

- It decide what to do by finding sequences of actions that lead to desirable states.



```
function SIMPLE-PROBLEM-SOLVING-AGENT(percept) returns an action
  inputs: percept, a percept
  static: seq, an action sequence, initially empty
          state, some description of the current world state
          goal, a goal, initially null
          problem, a problem formulation

  state ← UPDATE-STATE(state, percept)
  if seq is empty then do
      goal ← FORMULATE-GOAL(state)
      problem ← FORMULATE-PROBLEM(state, goal)
      seq ← SEARCH(problem)
  action ← FIRST(seq)
  seq ← REST(seq)
  return action
```

Search

Search: A problem-solving agent

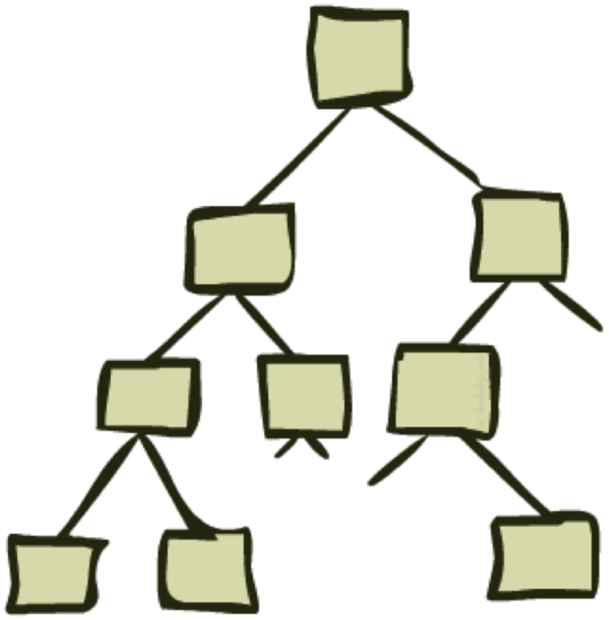
- Searching is a step-by-step procedure to solve a problem in a given search space.
- Search techniques are universal problem-solving methods.
- Rational agents or Problem-solving agents in AI mostly use a search strategy or algorithm to solve a specific problem and provide the best result.
- Problem-solving agents are a kind of goal-based agent.
- A search algorithm takes a problem as input and returns a solution in the form of an action sequence.

Search

Search: Terminology

The search methods are defined mostly on trees and graphs, so let’s discuss some terminology for these structures:

- A **tree** is made up of nodes and links (circles/square and lines) connected so that there are no loops (cycles). Nodes are sometimes referred to as vertices and links as edges.



Search

Search: Terminology

- A tree has a root node (where the tree "starts"). Every node except the root has a single parent (aka direct ancestor). More generally, an ancestor node is a node that can be reached by repeatedly going to a parent node.
- Each node (except the terminal (aka leaf) nodes) has one or more children (aka direct descendants). More generally, a descendant node is a node that can be reached by repeatedly going to a child node.

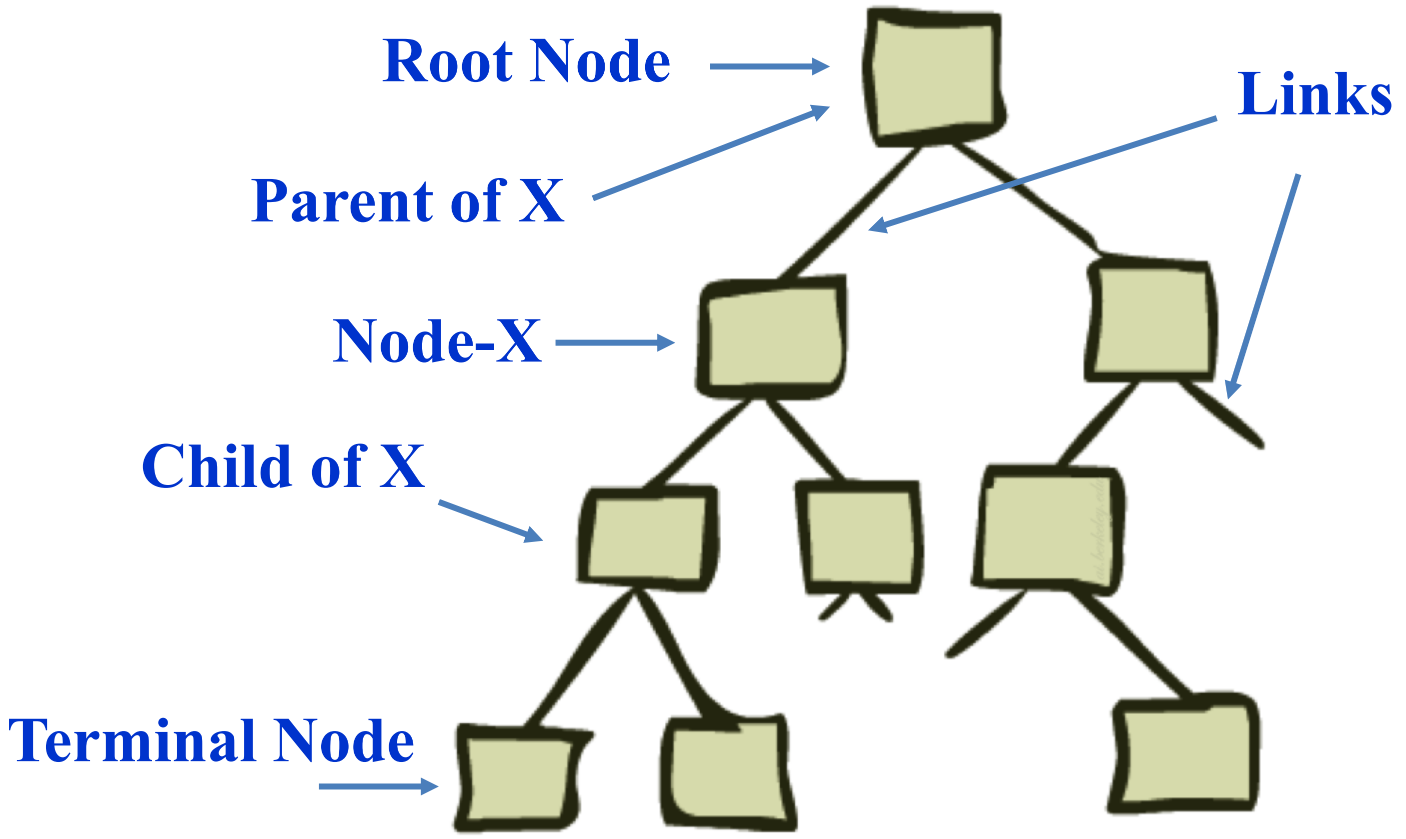
Search

Search: Terminology

- A tree has a root node (where the tree "starts"). Every node except the root has a single parent (aka direct ancestor). More generally, an ancestor node is a node that can be reached by repeatedly going to a parent node.
- Each node (except the terminal (aka leaf) nodes) has one or more children (aka direct descendants). More generally, a descendant node is a node that can be reached by repeatedly going to a child node.

Search

Search: Terminology



Search

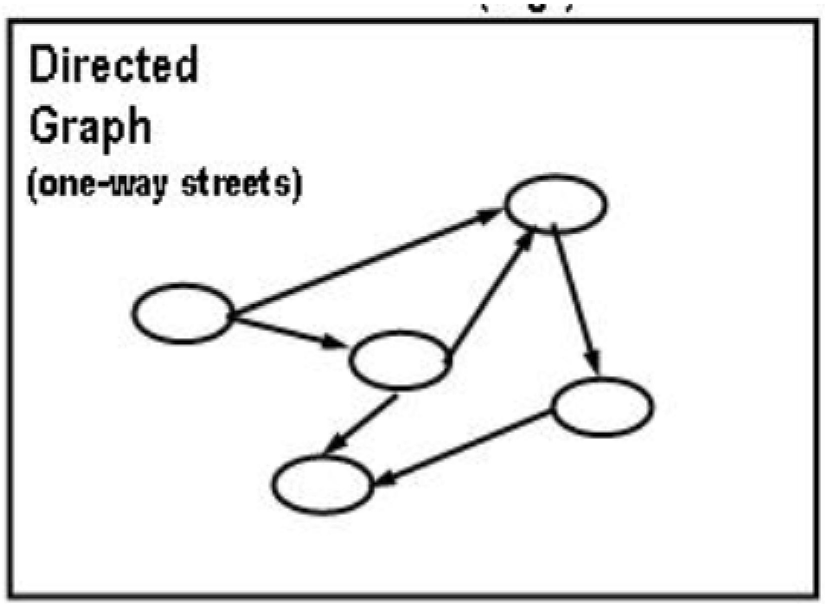
Search: Terminology

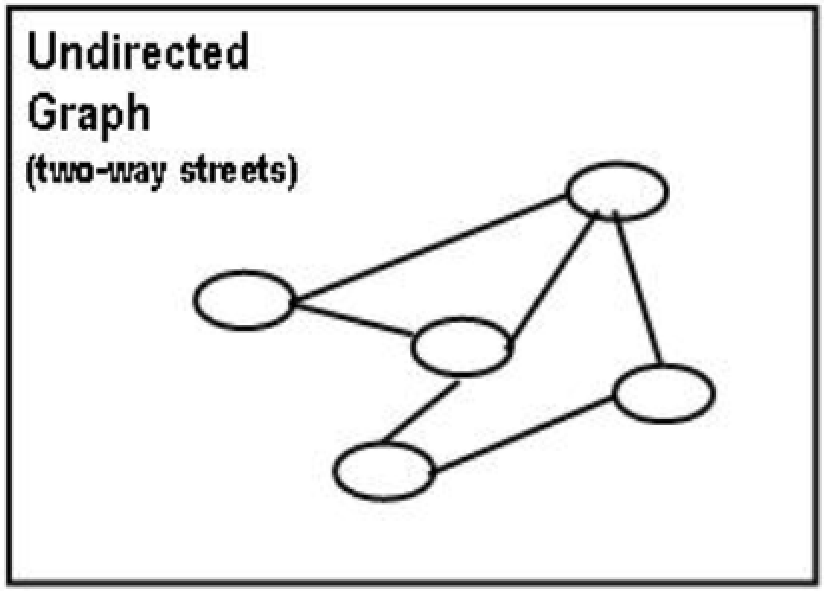
- A **graph** is also a set of nodes connected by links but where loops are allowed, and a node can have multiple parents. We have two kinds of graphs to deal with:
- **Directed graphs**, where the links have direction (akin to one-way streets).
- **Undirected graphs,** where the links go both ways. You can think of an undirected graph as shorthand for a graph with directed links going each way between connected nodes.

Search

Search: Terminology

- A **graph** is also a set of nodes connected by links but where loops are allowed and a node can have multiple parents. We have two kinds of graphs to deal with:





Search

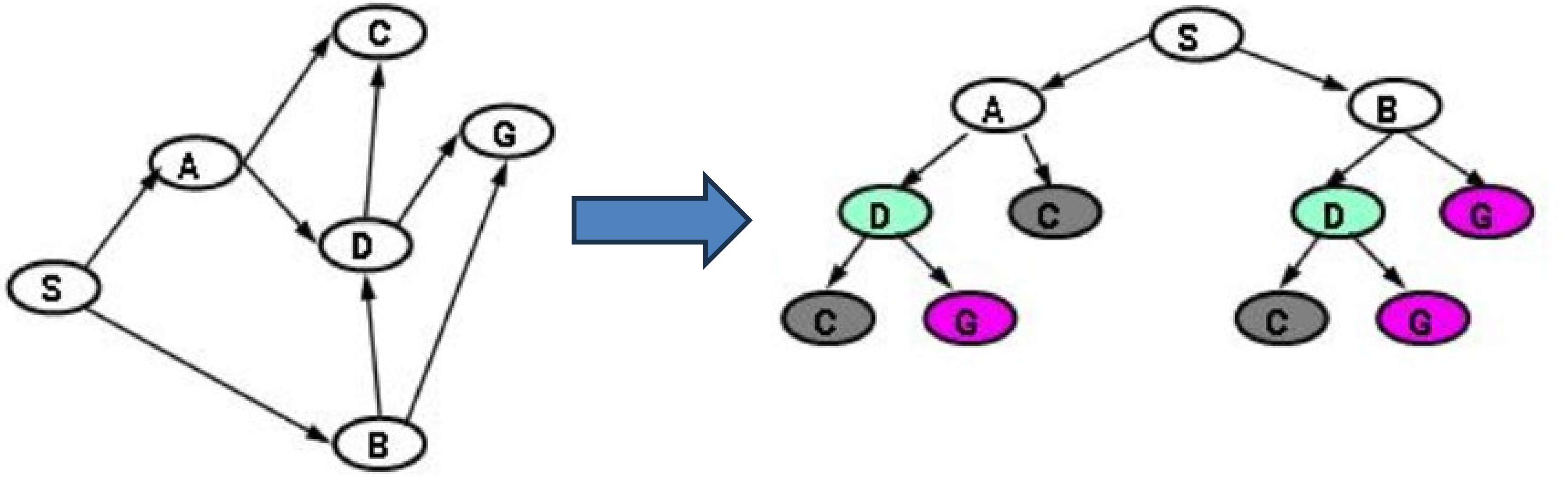
Search: Graph search vs Tree search

- Trees are a subclass of directed graphs, which don't have cycles and every node has a single parent.
- Cycles are bad for searching, since, obviously, you don't want to go round and round getting nowhere.
- A graph can be converted to an equivalent problem of searching a tree by doing two things: turning undirected links into two directed links; and, more importantly, making sure we never consider a path with a loop or, even better, by never visiting the same node twice.

Search

Search: Graph search vs Tree search

- Example: We have a graph with S as the start point, and our goal is to find a path to reach G.



Search

Search: As a problem-solving agent

One general approach to problem solving in AI is to reduce the problem to be solved to one of searching a graph. To use this approach, we must specify what are the **states**, the **actions** and the **goal test**.

Assumptions

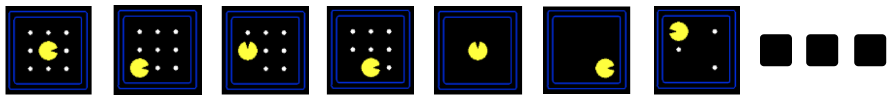
The actions are deterministic, that is, we know exactly the state after the action is performed.

The actions are discrete, so we don't have to represent what happens while the action is happening.

Search

What a search problem should have?

- A search problem consists of:
  - A state space $\mathcal{S}$

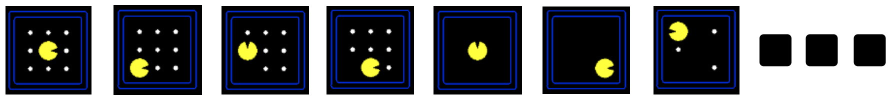


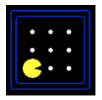
Search space or state space represents a set of possible solutions, which a system may have.

Search

What a search problem should have?

- A search problem consists of:
  - A state space $\mathcal{S}$
  - An initial state s_0



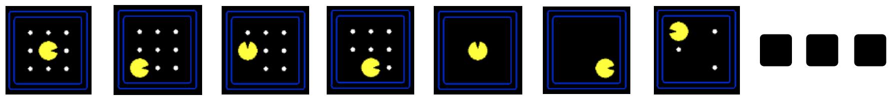


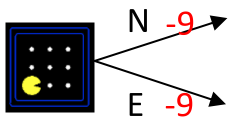
It is a state from where agent begins the search.

Search

What a search problem should have?

- A search problem consists of:
  - A state space $\mathcal{S}$
  - An initial state s_0
  - Actions $\mathcal{A}(s)$ in each state



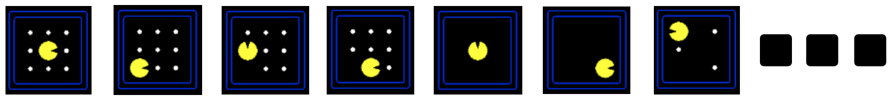


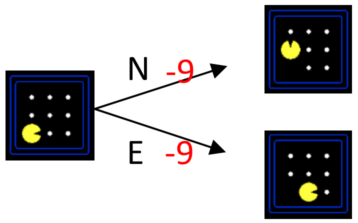
Actions gives the description of all the available actions to the agent.

Search

What a search problem should have?

- A search problem consists of:
  - A state space $\mathcal{S}$
  - An initial state s_0
  - Actions $\mathcal{A}(s)$ in each state
  - Transition model *Result*(s,a)



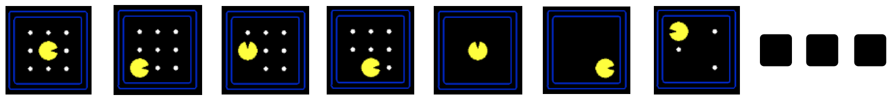


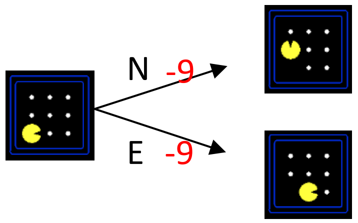
A description of what each action do, can be represented as a transition model.

Search

What a search problem should have?

- A search problem consists of:
  - A state space $\mathcal{S}$
  - An initial state s_0
  - Actions $\mathcal{A}(s)$ in each state
  - Transition model *Result*(s,a)
  - A goal test $G(s)$
    - S has no dots left



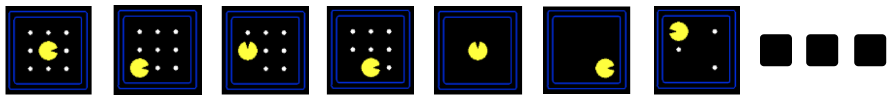


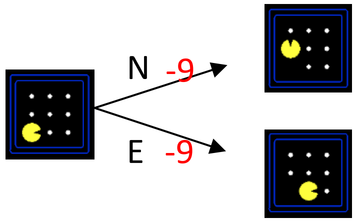
Goal test is a function which observe the current state and returns whether the goal state is achieved or not.

Search

What a search problem should have?

- A search problem consists of:
  - A state space $\mathcal{S}$
  - An initial state s_0
  - Actions $\mathcal{A}(s)$ in each state
  - Transition model *Result*(s,a)
  - A goal test $G(s)$
    - S has no dots left
  - Action cost $c(s,a,s')$
    - +1 per step; -10 food; -500 win; +500 die; -200 eat ghost



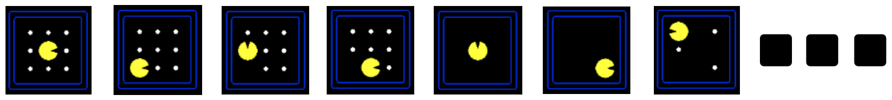


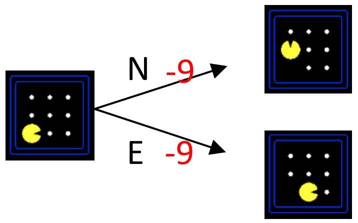
Action/path cost is a function which assigns a numeric cost to each action/path.

Search

What a search problem should have?

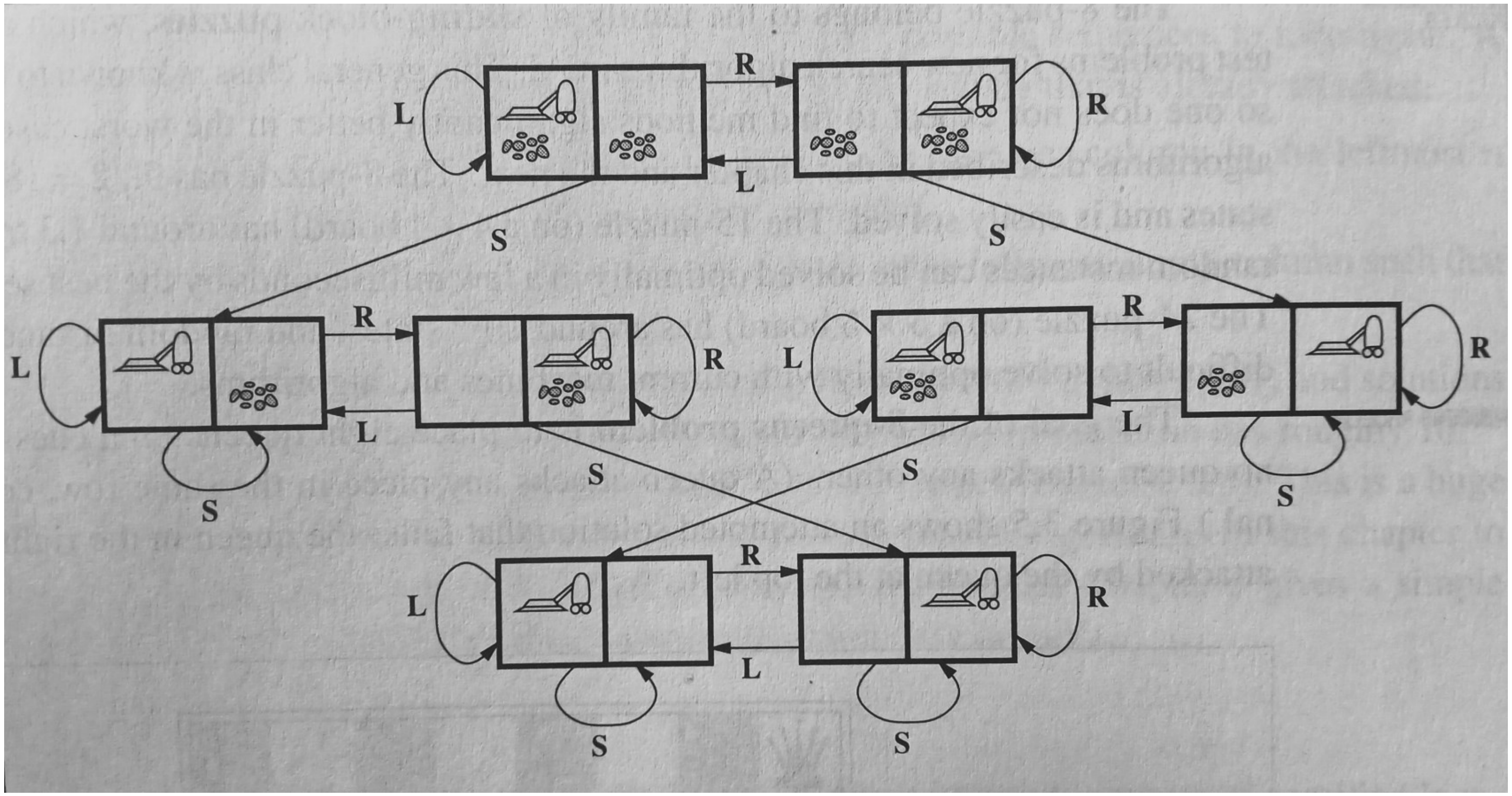
- A search problem consists of:
  - A state space $\mathcal{S}$
  - An initial state s_0
  - Actions $\mathcal{A}(s)$ in each state
  - Transition model $Result(s,a)$
  - A goal test $G(s)$
    - S has no dots left
  - Action cost $c(s,a,s')$
    - +1 per step; -10 food; -500 win; +500 die; -200 eat ghost
- A solution is an action sequence that reaches a goal state
- An optimal solution has least cost among all solutions





Search

Example of Problem State-space



Search

Search: Key point to remember

An important distinction that will help us keep things straight is that between a state and a search node.

- A **state** is an arrangement of the real world (or at least our model of it). We assume that there is an underlying "real" state graph that we are searching (although it might not be explicitly represented in the computer; it may be implicitly defined by the actions).
- We assume that you can arrive at the same real-world state by multiple routes, that is, by different sequences of actions.

Search

Search: Key point to remember

An important distinction that will help us keep things straight is that between a state and a search node.

- A **search node**, on the other hand, is a data structure in the search algorithm, which constructs an explicit tree of nodes while searching.
- Each node refers to some state (of real-world), but not uniquely (many nodes may refer to the same node).
- A node also corresponds to a path from the start state to the state associated with the node. This follows from the fact that the search algorithm is generating a tree. So, if we return a node, we're returning a path.