CISS450 Lecture 2: Agents

Yihsiang Liow

August 30, 2020

Table of contents I

Agenda

- Agents
- Readings: AIMA3 Chapter 2, AIMA4 Chapter 2.
- WARNING: The algorithms are presented as pseudocode.

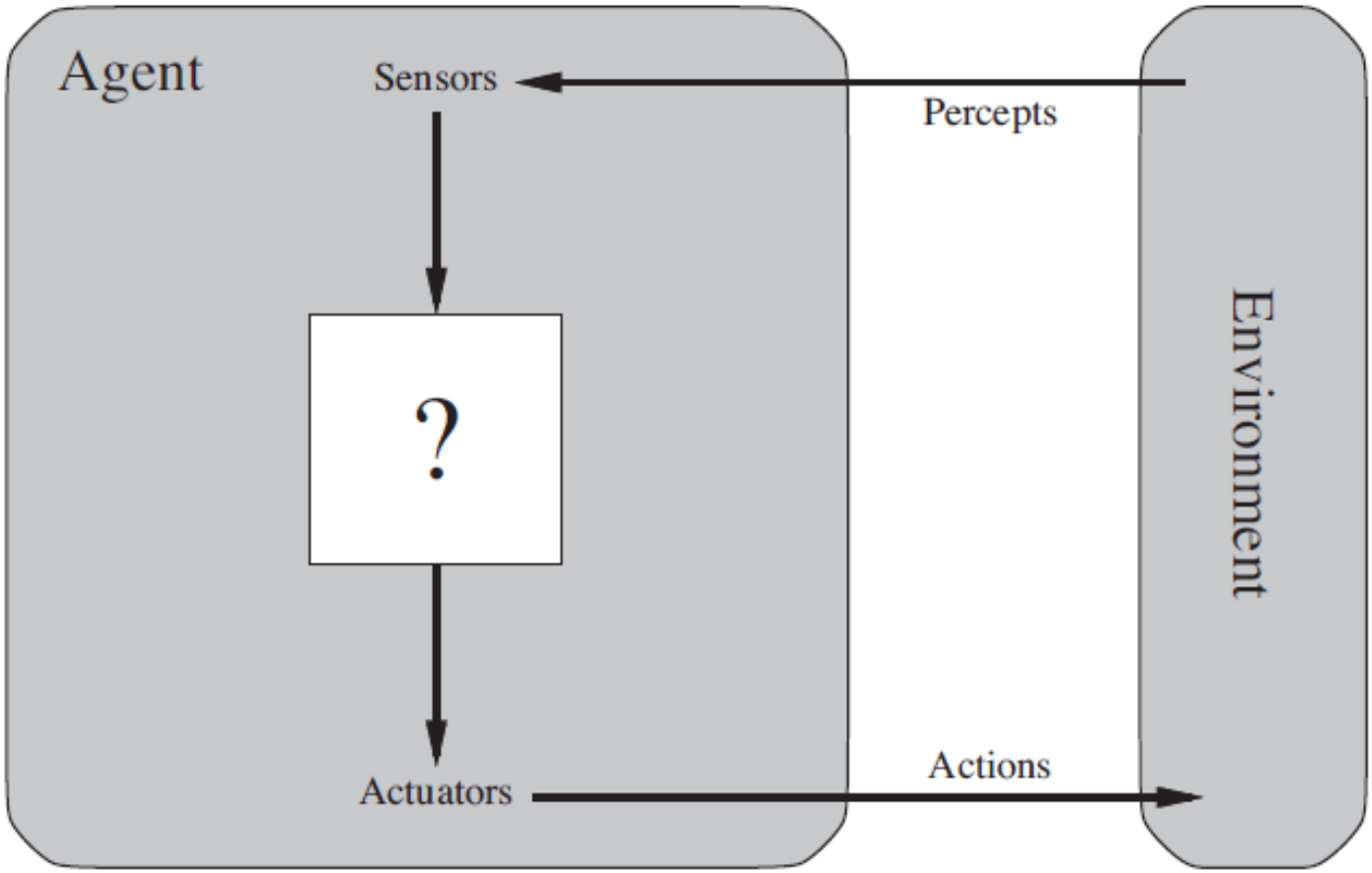
Agents I

- Agent = (informally) anything that sees and acts.
- **Agent** = (more formally) anything that senses the environment through sensors and acts on the environment through actuators
- The sensory data (input) is called **percept**.
- The **percept sequence** or **percept history** is the complete history of all percepts. (The agent might not save the whole percept sequence.)
- Example. Sensors: Camera, touch sensor, radar, etc.
- Example. Percepts: image data from camera, binary data from touch sensor, distance and direction to object from radar/lidar, etc.

Agents II

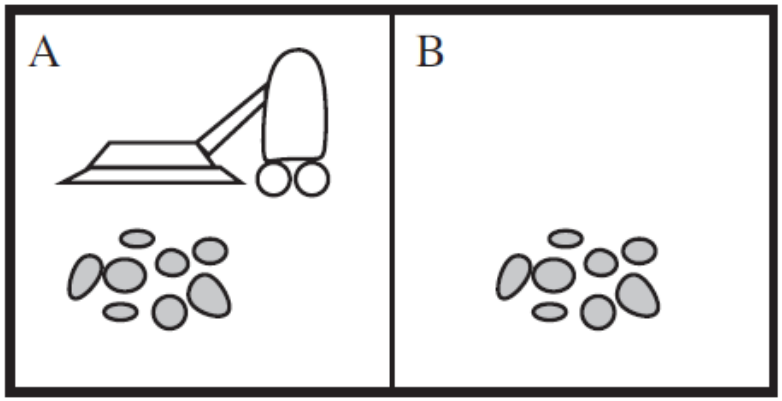
- **Actuators** changes agent in the environment
  - Actuator can turn the steering wheel of a car
  - Actuator can turn the camera
  - Etc.
- Some books differentiate between actuator and **effectors** -– I'll be vague since we probably won't have time to get into robotics or software simulation of robotics.
- The bottomline:
  - Wheel: effector
  - Motor: actuator

Agents III



Robot vacuum cleaner I

- Robot vacuum cleaner and an environment:



- Possible percepts: [A, Clean], [A, Dirty], [B, Clean], [B, Dirty]. In other words, assume the robot can sense which room it is in and it can also sense if the room is clean or dirty.

Robot vacuum cleaner II

- Possible actions: Right, Suck, Left. In other words, the robot can move right, move left, and suck.
  - Note: If robot is at the left room, moving left does not do anything. If the robot is at the right room, moving right does not do anything.

Nature of environments I

- **Fully/partially observable**:
  - Are agent's sensors capable of sensing all percepts relevant to maximizing performance?
- **Single/multi-agent**:
  - Are there other agents around?
  - Are the other agents friends (cooperative) or enemies (competitive)?
  - Can an agent communicate with another?
- **Deterministic/stochastic**:
  - Does the environment change in a probabilistic manner?
  - Example: robot's wheel come off
  - Example: dirt appears randomly in robot vacuum world

Nature of environments II

- **Episodic vs sequential**:
  - **Episodic task environment**: Task is made up of subtasks which are independent
  - Example: Cleaning the world means cleaning room A and room B. Cleaning room A won't affect room B.
  - **Sequential task environment**: Task is also made up of subtasks, but a subtask can affect environment and therefore change later subtasks.
  - Example: In chess, making a deliberate queen sacrifice now could be the first step in a sequence of moves leading up to a massive attack/checkmate later. The sequence of moves are related/dependent.

Nature of environments III

- **Static vs dynamic**: An environment is static is it does not change while waiting for agent to make an action.
  - Example: Autonomous driving operates in a dynamic environment.

Nature of environments IV

- **Discrete vs continuous**: Refers to time, state of environment, percepts, and actions.
  - Example: In chess, percepts and actions are discrete.
  - Example: Autonomous driving (a vehicle) has continuous state, time, action. Visual percept is discrete (pixels) but is usually treated as continuous.

Nature of environments V

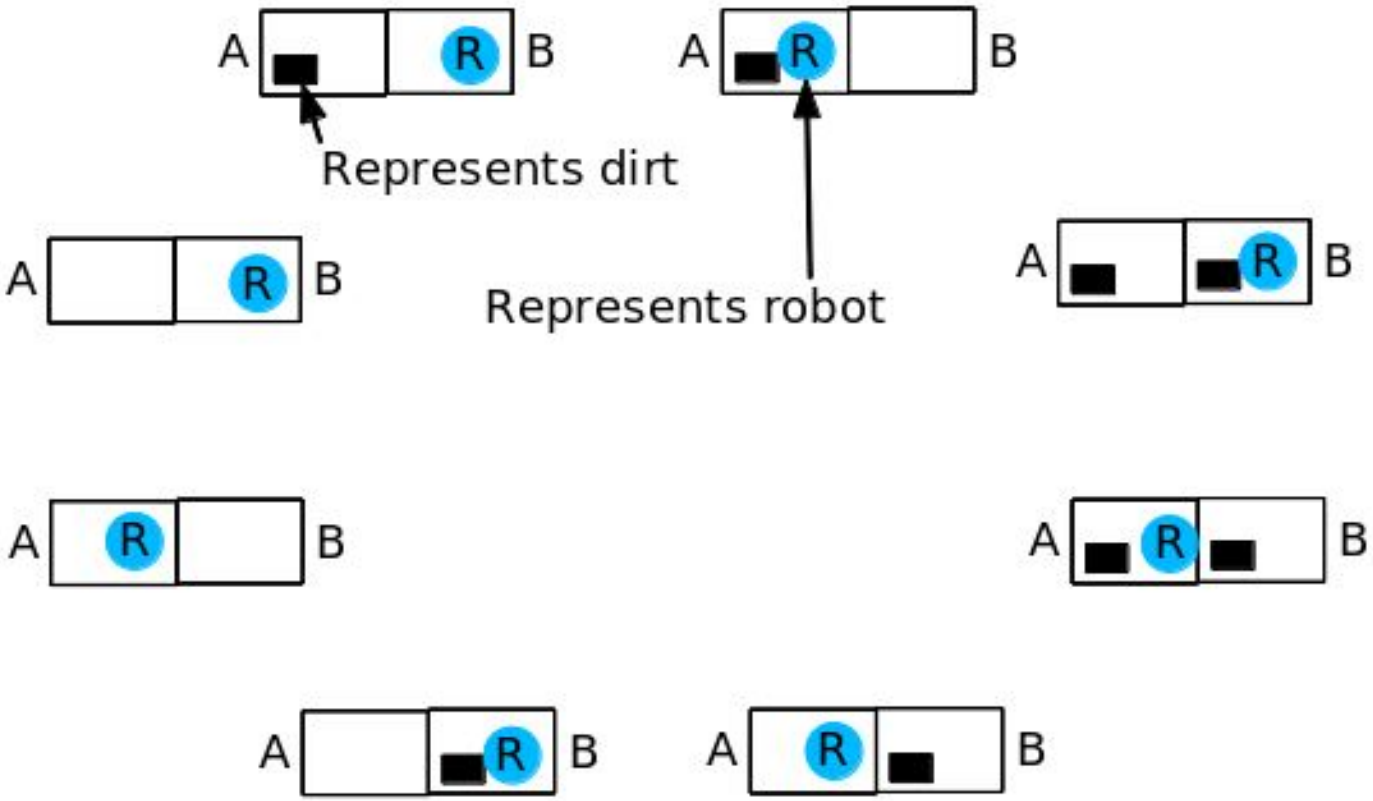
- **Known vs unknown**: Whether the rules of how the environment works is known to the agent or not. (This is more about the agent than the environment.)
  - Example: A software agent plays Galaxian and has control over the keyboard, has access to the screen pixels, and has access to its score, knows that it has to maximize the score, but does not understand how scores are computed or the interaction between lasers and aliens and itself.

State space I

- **State** = description of the world/environment
- Example: Robot vacuum problem. Each state is described by location of robot, state of room A, state of room B. Location of robot is A or B. State of room A (or B) is clean or dirty.

State space II

- Example. State space for vacuum cleaner robot program:

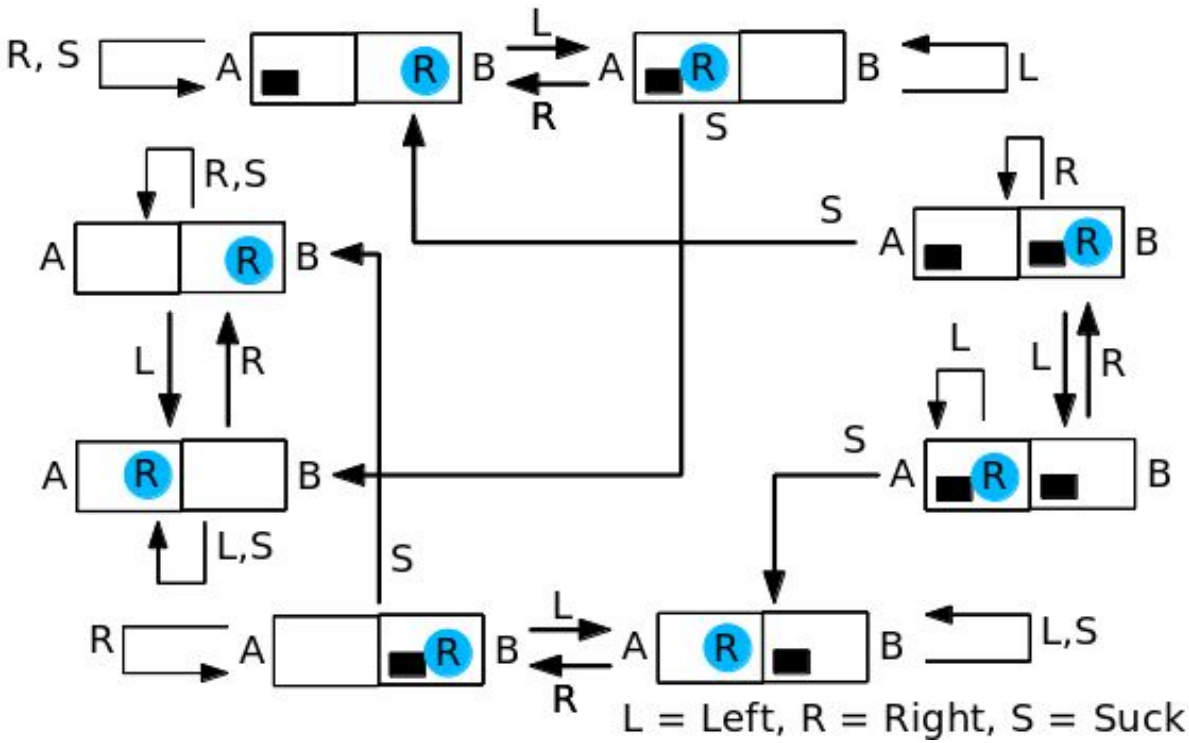


State space III

- When an agent sends an action to its actuator, it will cause the state of the environment to change.
- Including actions, the state space becomes a graph – the **state graph**. Arrows are actions labeled with the action.
- The state graph is like a DFA (deterministic finite state automata) except that the start state is not shown since it depends on where the agent begins.

State space IV

- Example. State graph for the vacuum cleaner robot program: Actions: L = left, R = right, S = suck.

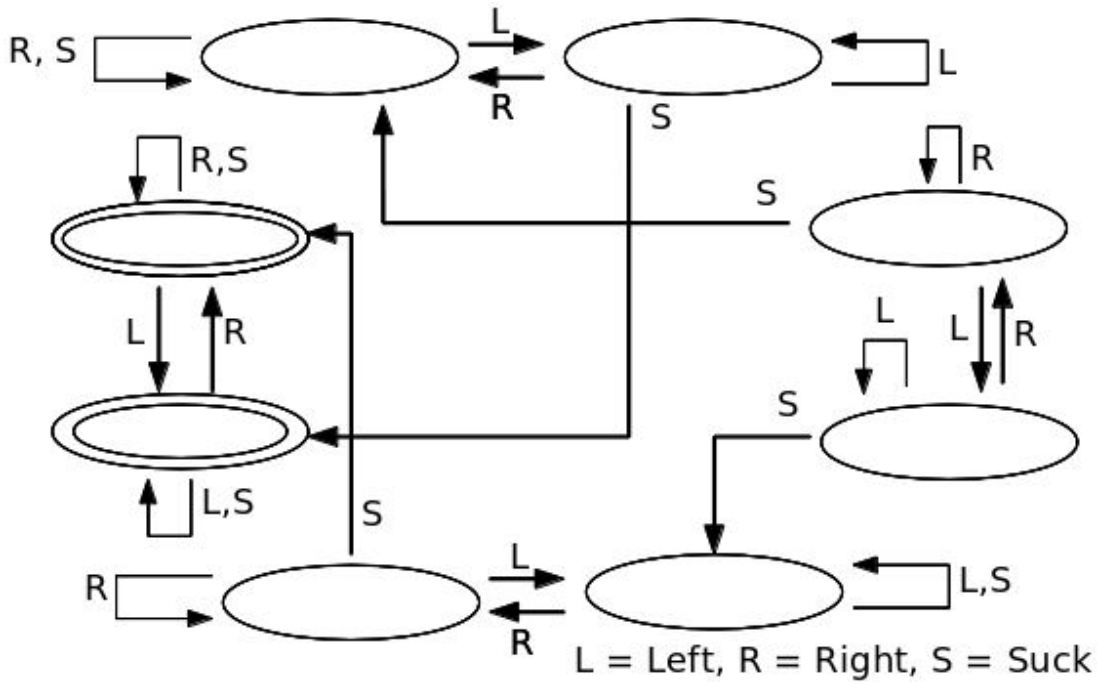


State space V

- Note that we are seeing the whole state space and how each action at a particular state changes the state. In real life, it's usually impossible to see whole state space because it's too big.

State space VI

- You can also think of the above state graph as a general graph where you can move from one node to another and the goal is to reach certain special nodes (goal states).



State space VII

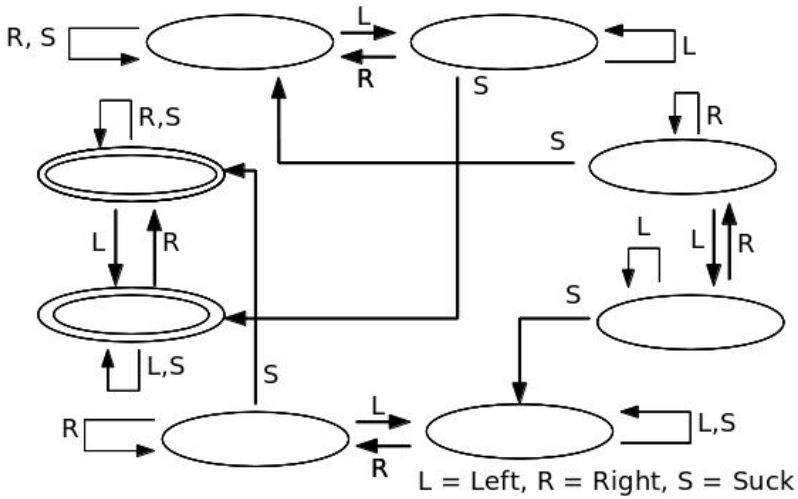
- The action the agent chooses is computed from percept it receives from the environment (through sensors).
- Example: Robot vacuum problem. Percept = [A, Clean], [A, Dirty], [B, Clean], or [B, Dirty]. If percept = [A, Clean], it means that the location sensor says that the robot is in room A and the dirt sensor syas that there's dirty at the current robot's location.

State space VIII

- An agent's actions will change the change the environment from one state to another.
- So the question is what are the desirable state(s) ... and how should an agent choose (i.e. compute) the right actions to get to the desirable state(s) ... see next section.

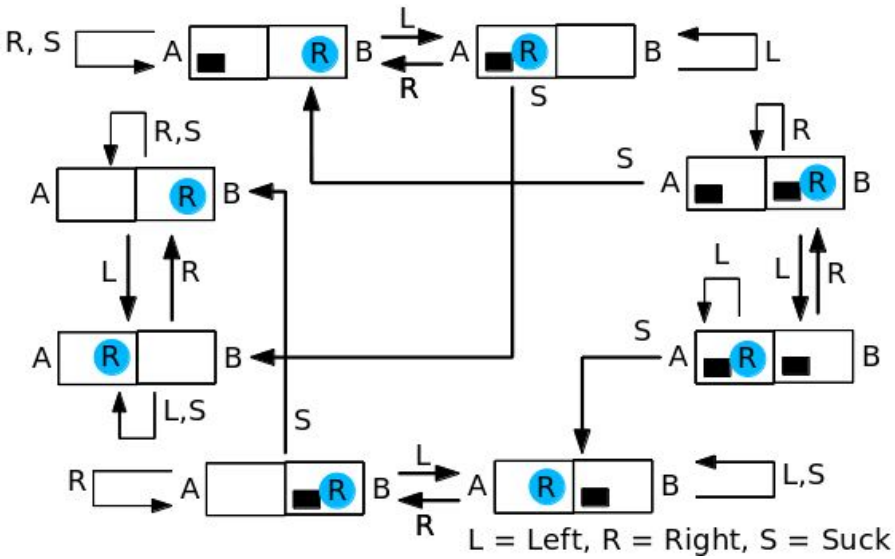
State space IX

- Exercise. Important exercise! View this as an abstract state graph (forget about the fact that it's the vacuum cleaner robot problem): Find a sequence of actions so that no matter what state you start with, you always end up in one of the two states drawn with double-edges boundary. Does your strategy work if I change the shape of the state graph?



State space X

- Exercise. Now view the state graph above as the vacuum cleaner robot problem. Again, find a sequence of actions so that no matter what state you start with, you always end up in one of the two leftmost states, i.e., the states where room A and B are clean.



Rational agent I

- **Rational agent** = (informally) an agent that acts in a rational way.
- "Rational"??? Need to have some form of performance measure to indicate success.
- Formally: An agent is **rational** if for each percept sequence, the selected action maximizes some performance measure.
- The performance measure is a performance measure on the state of the environment achieved by the agent.

Rational agent II

- A rational agent selects action based on an **agent program** – the thinking part of the agent.
- The selection of a rational action depends on:
  - Agent's performance measure
  - Agent's prior knowledge of environment
  - Agent's possible action
  - Agent's percept sequence up to this point in time

Rational agent III

- Example. For the vacuum cleaner robot problem, one possible performance measure is to have room A and B both clean. If each action has a cost (usage of electricity), then another measure is to achieve the above using the least amount of energy. If there's a Stop action that shuts down the robot, then Stop should be executed once all rooms are clean.
- In real life, depending on the role of the agent, rationality (i.e., maximizing performance) can be very difficult or even impossible. More practical to maximize expected (i.e., average) performance over time.

Omniscience I

- **Omniscience**: Agent knows the *actual* resulting state of the environment based on its action
- Formally: An agent is **omniscient** if
  - The agent knows the current state s of the environment.
  - If the agent computes the resulting environment state, s', when an action a is applied to the current state s and the agent sends action a to actuators so that the actual resulting environment state is s'', then s' and s'' are the same.
- Example: A chess playing agent is omniscient.

Omniscience II

- In the real world omniscience is usually impossible.
  - There's a limit to amount of percept data available because of sensor limitation. Example: Robot might not have view sensors in all possible directions.
  - Actuators/effectors does not work perfectly. Example: An actuator rotating a robot's wheel by $360°$ might actually turn $359.5°$
  - Imperfection in the physical world. Example: A flat road that a self-driving car is on is not perfectly flat.
  - There are other forces that affect change or acting on the environment. Example: Other agents, physics, etc.

Autonomy I

- Assumptions about the environment might be incomplete or incorrect. The environment might change.
- An agent is **autonomous** if it has the ability to learn and compensate for the incomplete/incorrect/changing information about the environment.

Structure of Agent I

- Agent = architecture + agent program
- **Architecture** (in AIMA) = the physical device including sensors + actuators
- **Agent program**: the part of the agent that selects the action based on the percepts (for us, it's the software)
- View the agent program as a function/procedure that is called for each new percept and returns an action.
- The following TABLE-DRIVEN-AGENT (ideal) remembers the complete percept history in memory.

Structure of Agent II

```
ALGORITHM   TABLE-DRIVEN-AGENT:
INPUT:      percept
OUTPUT:     action
PERSISTENT: percepts - initially empty
            table - fully specified, a table lookup
                    where every percept sequence is a
                    key and the value is an action

append percept to percepts
action = lookup table using percepts as key
return action
```

Structure of Agent I

- In the pseudocode, `persistent` means that data is always around. Same as static in C++.
- In terms of software objects, this means that a table-drive agent, `x`, has an instance variables `x.percepts` and `x.table`.

```
class TableDrivenAgent:
    def __init__(self):
        self.percepts = []
        self.table = {...}
    def compute_action(self, percept)
        self.percepts.append(percept)
        action = self.table[self.percepts]
        return action
```

Structure of Agent I

- If the percept looks like `['A', True]` where the robot is in room A and room A is dirty, then the table would look like

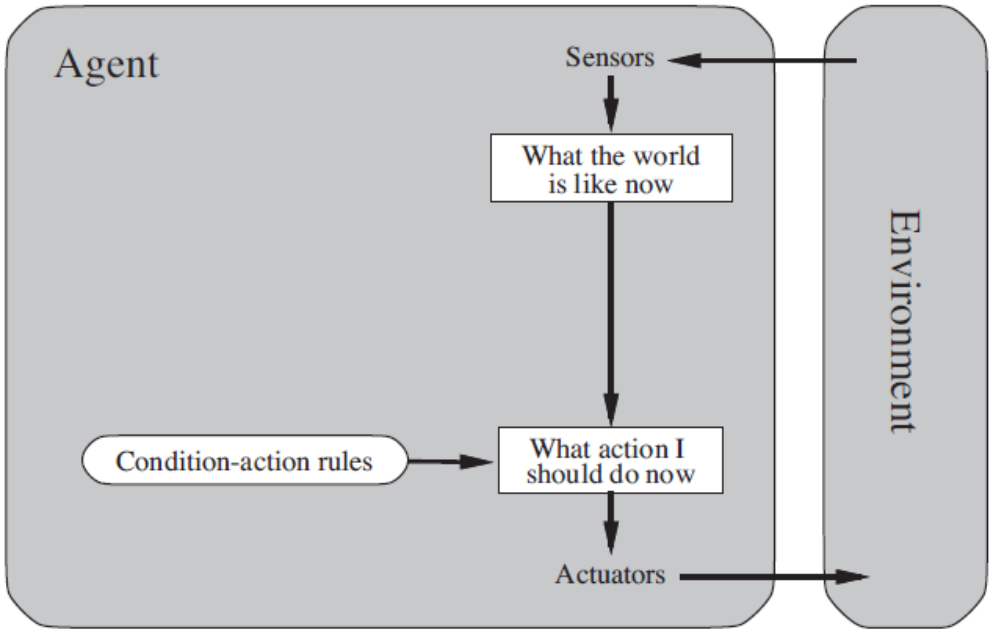
```
self.table = {[['A', True]]: 'S',
              [['A', False]]: ...,
              [['B', True]]: ...,
              [['B', False]]: ...,
              [['A', True], ['A', True]]: ...,
              [['A', True], ['A', False]]: ...,
              [['A', True], ['B', True]]: ...,
              [['A', True], ['B', False]]: ...,
              [['A', False], ['A', True]]: ...,
              [['A', False], ['A', False]]: ...,
              [['A', False], ['B', True]]: ...,
              [['A', False], ['B', False]]: ...,
              ...
              }
```

Agent Programs I

- Some basic types of agents:
  - Simple reflex agents
  - Model-based agents
  - Goal-based agents
  - Utility-based agents
  - Learning agents

Simple Reflex Agent I

- **Simple reflex agent**: Action based on a single new percept, not based on percept history.
  - Forgetful and can go into an infinite loop.



Simple Reflex Agent II

- Implementation with condition-action, i.e., as a stack of if-else statement.
  - Example. Autonomous car: if (car-in-front-brakes), do (brake).
- Implementation as rule-based system:
  1. Interpreter translates percept to target state, and then
  2. Run through collection of (state, rule), find first (state, rule) whose state is target state from 1.
  3. Get action from rule from 2.

  Why do it this way? The condition-rules method can be difficult to maintain for large systems, with lots of deeply nested substructures.

Simple Reflex Agent III

```
ALGORITHM:  SIMPLE-REFLEX-AGENT:
INPUT:      percept
OUTPUT:     action
PERSISTENT: rules

state = INTERPRET-INPUT(percept)
rule = RULE-MATCH(state, rules)
action = rule.action
return action
```

Simple Reflex Agent IV

- Example: Vacuum Cleaner Robot Problem – condition-action pairs. Let percept be (location, status).

```
if status is Dirty:
    action = Suck
else:
    if location is A:
        action = Right
    else:
        action = Left
```

The robot does not remember if it was in Room A earlier. Therefore if there's an action Stop to stop the robot, it would not know when to use it.

Simple Reflex Agent V

- Example: Vacuum Clean Robot Problem – rules.

```
ALGORITHM INTERPRET-INPUT(percept):
    location, status = percept
    if location==A: return [A, status, None]
    else: return [B, None, status]
rules = [[[A, Dirty, None], DIRTY_RULE],
         [[B, None, Dirty], DIRTY_RULE],
         [[A, Clean, None], A_CLEAN_RULE],
         [[B, None, Clean], B_CLEAN_RULE]]
function RULE-MATCH(target_state, rules):
    for [state,rule] in rules:
        if state == target_state: return rule
```

  - DIRTY_RULE, A_CLEAN_RULE, B_CLEAN_RULE are rule objects with DIRTY_RULE.action=Suck, A_CLEAN.action=Right, B_CLEAN.action=Left.

Simple Reflex Agent VI

- Note that the state computed by INTERPRET-INPUT is not a complete environment state. [A, Dirty, None] means that the robot is in room A and room A is dirty, but nothing is known about room B.

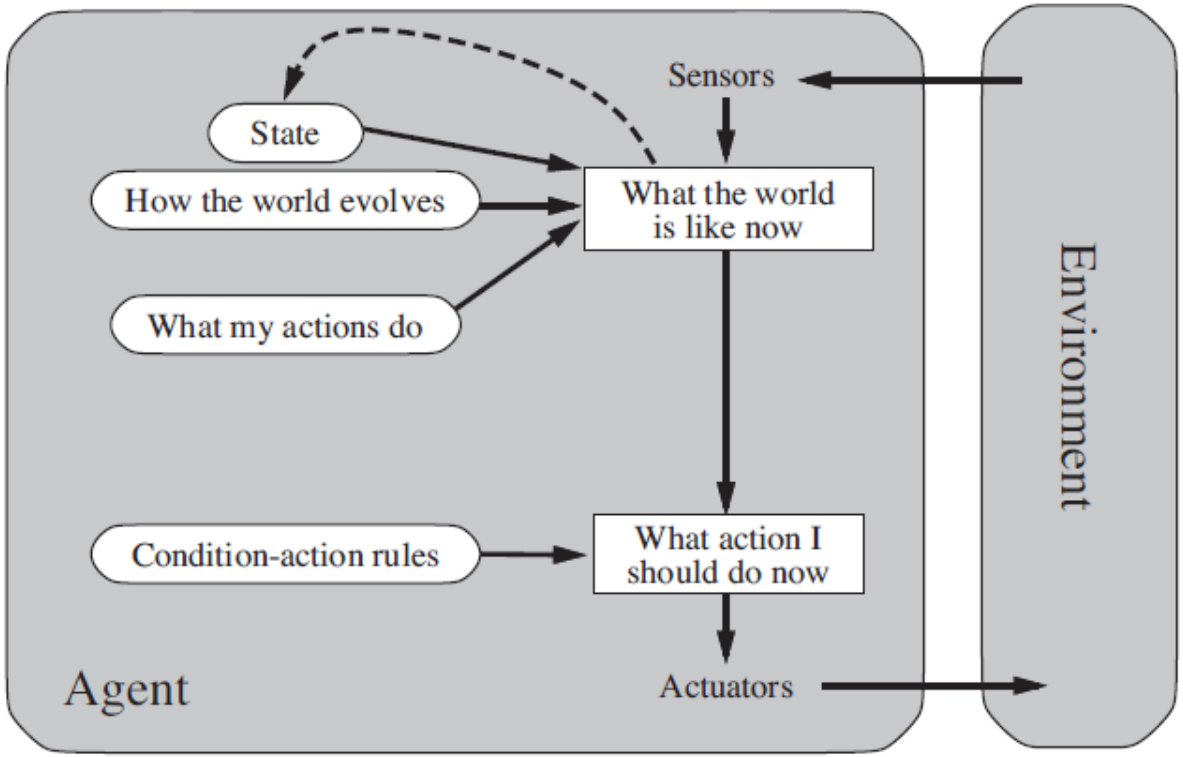
Model-based Reflex Agent I

- **Model** = Description of how world works.
  - An agent can build its own internal representation of state to model the state (possibly partial and approximation) of the actual external state of environment.
  - Watch out: There are two states, internal and external. Watch the context.
  - Why? Partial observability problem. Because temporarily some objects in the environment might not be observable by sensors. But might be observed earlier.
    - Example: Autonomous vehicle. Pedestrian might be temporarily blocked from view.
  - Agent can compute internal state based on (1) percepts and (2) previous state and model on how the world works.

Model-based Reflex Agent II

- Note: In the algorithm below (from AIMA),
  - The agent will probably also need to keep track of the time difference between state updates.
  - If there's enough memory, storing more than one state might be useful.
- **Model-based reflex agent** = Might be able to figure out (to some extent) the state of the part of the world that is current not observable. The agent does so using a model.

Model-based Reflex Agent III



Model-based Reflex Agent IV

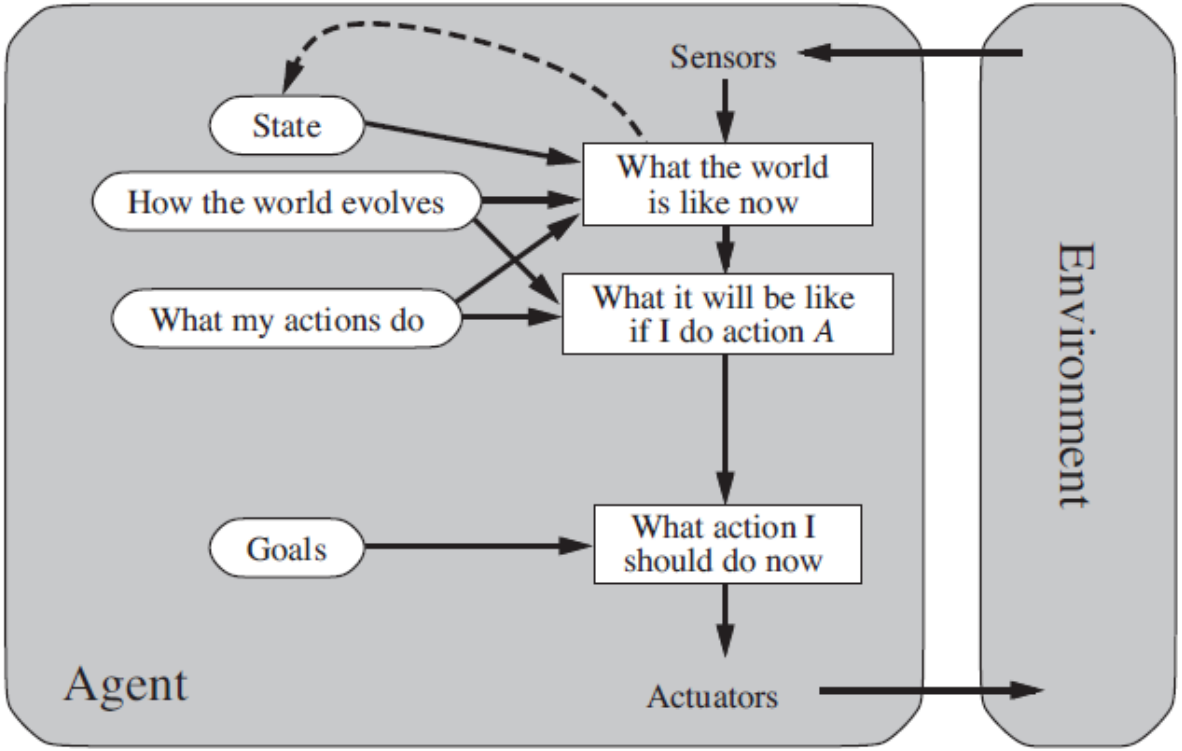
```
ALGORITHM:  MODEL-BASED-REFLEX-AGENT
INPUT:      percept
OUTPUT:     action
PERSISTENT: state: agent's current conception of world state
            model: description of how next state depends
                   on current state + action
            rules: set of condition-action rules
            action: last action (initially none)

state = UPDATE-STATE(state, action, percept, model)
rule = RULE-MATCH(state, rules)
action = rule.action
return action
```

Goal-based Agent I

- **Goal-based agent**
  - Is a model-based reflex agent
  - Knows if a state is a goal state or not. There can be more than one actual goal state.
  - Knows what are the possible actions at current state.
  - Test each action by applying each action to state and compute resulting state and test if resulting state is a goal state. Action is selected accordingly to achieve goal state.
  - Can involve search and planning. Search means looking for goal state. (Planning is more complex.)

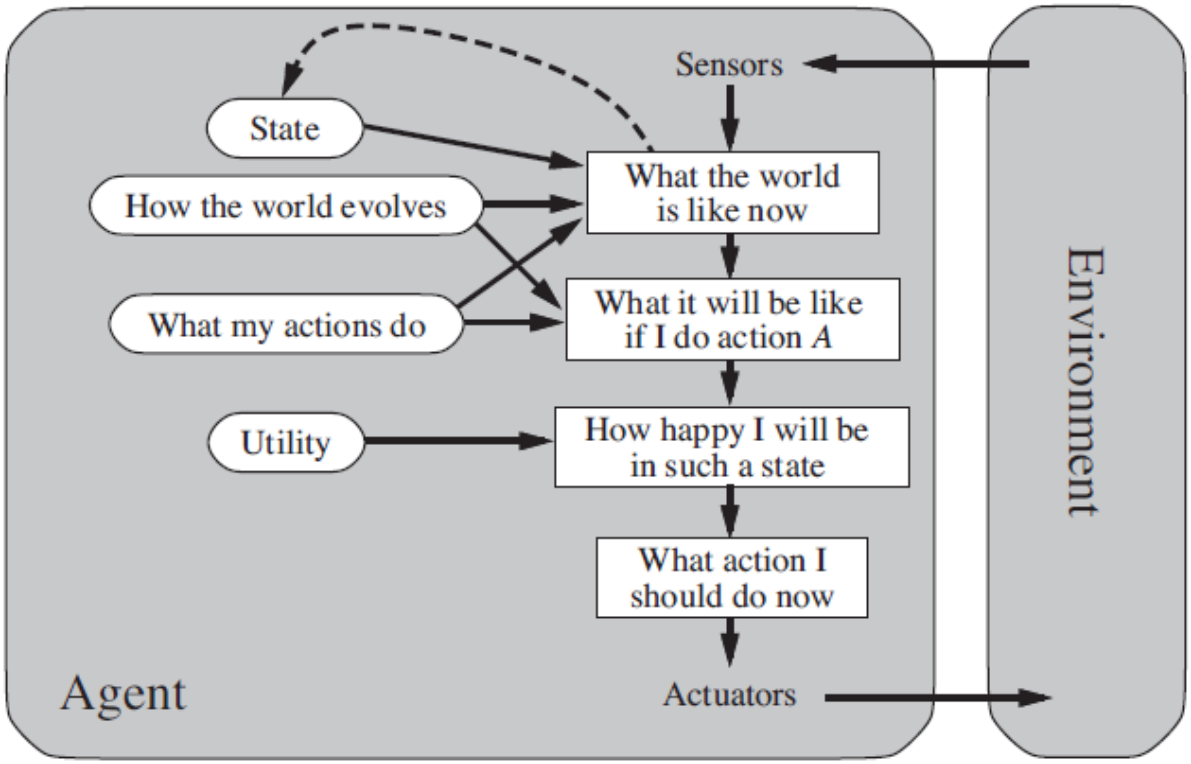
Goal-based Agent II



Utility-based Agent I

- Note that goal-based agents can only tell if a state is a goal state or not a goal state. (There might be more than one.)
- Utility = measure quality of states
- **Utility-based agent** =
  - There are several (potentially conflicting) goals
  - Agent tries to maximize the expected utility by computing a weighted average of utility of the various goals when an action is carried out. (Expected means "average" in the probabilistic sense.)
- Example: A self-driving car has reach destination, obey traffic rules, not kill (!), save fuel. When are these goals in conflict?

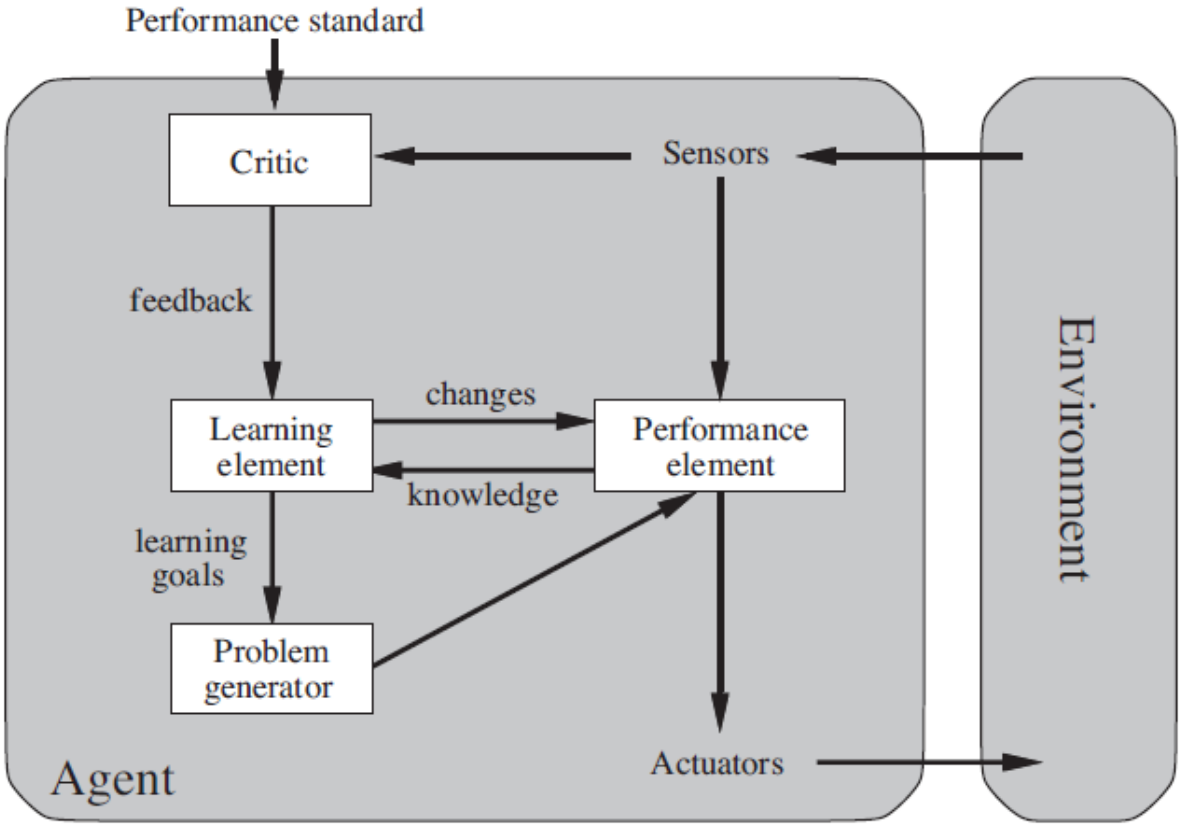
Utility-based Agent II



Learning Agent I

- Agent improves with time.

Learning Agent II



Exercises I

- Describe the environment and state the type of agent:
  - Chess playing program
  - Pharmacy robot
    `https://www.youtube.com/watch?v=oumlYbwfAsI`:
  - A vacuum cleaner robot that systematically cleans a living room and can plug itself to power when necessary.