Agent programs?

The agent programs all have the **same skeleton**:
they take the current percept as input from the sensors and return an action to the actuator, Notice the difference between the **agent program**, which takes the current percept as input, and the **agent function**, which takes the entire percept history. The agent program takes just the current percept as input because nothing more is available from the environment; if the agent's actions depend on the entire percept sequence, the agent will have to remember the percepts.

Function TABLE-DRIVEN_AGENT(*percept*) **returns** an action

static: *percepts*, a sequence initially empty
table, a table of actions, indexed by percept sequence

append *percept* to the end of *percepts*

action ← LOOKUP(*percepts*, *table*)

return *action*

Figure 1.8 The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns **an** action each time.

Drawbacks:?

- **Table lookup** of percept-action pairs defining all possible condition-action rules necessary to interact in an environment
- **Too big** to generate and to store (Chess has about 10^{120} states, for example)
- **No knowledge** of non-perceptual(cannot perceive) parts of the current state
- **Not adaptive** to changes in the environment; requires entire table to be updated if changes occur
- **Looping**: Can't make actions conditional
- Take a long time to build the table
- No autonomy
- Even with learning, need a long time to learn the table entries

Some Agent Types

- **Table-driven agents**
  - use a percept sequence/action table in memory to find the next action. They are implemented by a (large) **lookup table**.
- **Simple reflex agents**
  - are based on **condition-action rules**, implemented with an appropriate production system. They are **stateless devices which do not have memory** of past world states.
- **Agents with memory**
  - have **internal state**, which is used to keep track of past states of the world.
- **Agents with goals**
  - are agents that, in addition to state information, have **goal information** that describes desirable situations. Agents of this kind take future events into consideration.
- **Utility-based agents**
  - base their decisions on **classic axiomatic (Evident without proof or**

argument) utility theory in order to act rationally.

Simple Reflex Agent

The simplest kind of agent is the **simple reflex agent.** These agents select actions on the basis of the *current* percept, ignoring the rest of the percept history. For example, the vacuum agent whose agent function is tabulated in Figure 1.10 is a simple reflex agent, because its decision is based only on the current location and on whether that contains dirt.

- Select action on the basis of *only the current* percept.
  E.g. the vacuum-agent
- Large reduction in possible percept/action situations(next page).
- Implemented through *condition-action rules*

If dirty then suck

A Simple Reflex Agent: Schema

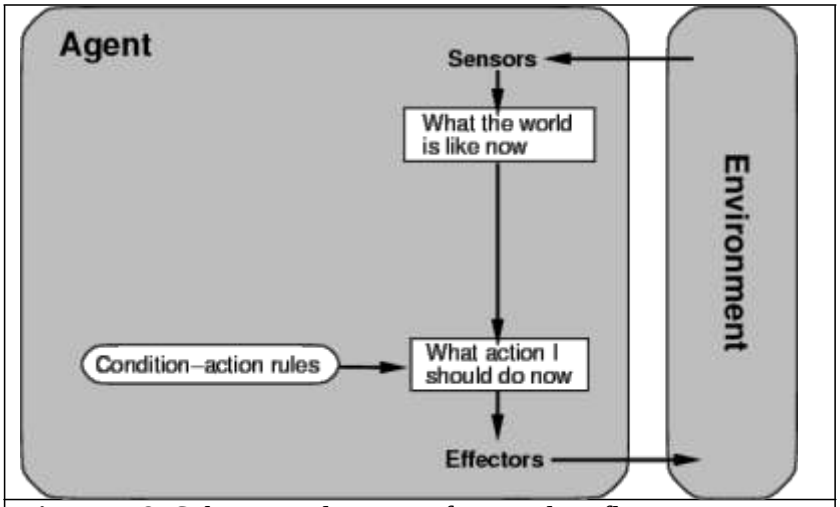

Figure 1.9 Schematic diagram of a simple reflex agent.

function SIMPLE-REFLEX-AGENT(*percept*) **returns** an action

static: *rules*, a set of condition-action rules
state ← INTERPRET-INPUT(*percept*)
rule ← RULE-MATCH(*state*, *rule*)
action ← RULE-ACTION[*rule*]
return *action*

Figure 1.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

function REFLEX-VACUUM-AGENT ([*location, status*]) return an
action if *status* == *Dirty* then return *Suck*
else if *location* == *A* then return *Right*
else if *location* == *B* then return *Left*

Figure 1.11 The agent program for a simple reflex agent in the two-state vacuum environment. This program implements the agent function tabulated in the figure 1.4.

- **Characteristics**
- Only works if the environment is fully observable.
- Lacking history, easily get stuck in infinite loops
- One solution is to randomize actions

Model-based reflex agents

The most effective way to handle partial observability is for the agent to *keep track of the part of the world it can't see now*. That is, the agent should maintain some sort of **internal state** that depends on the percept history and thereby reflects at least some of the unobserved aspects of the current state. Updating this internal state information as time goes by requires two kinds of knowledge to be encoded in the agent program. First, we need some information about how the world evolves independently of the agent-for example, that an overtaking car generally will be closer behind than it was a moment ago. Second, we need some information about how the agent's own actions affect the world-for example, that when the agent turns the steering wheel clockwise, the car turns to the right or that after driving for five minutes northbound on the freeway one is usually about five miles north of where one was five minutes ago. This knowledge about "how the world working - whether implemented in simple Boolean circuits or in complete scientific theories-is called a **model** of the world. An agent that uses such a MODEL-BASED model is called a **model-based agent.**

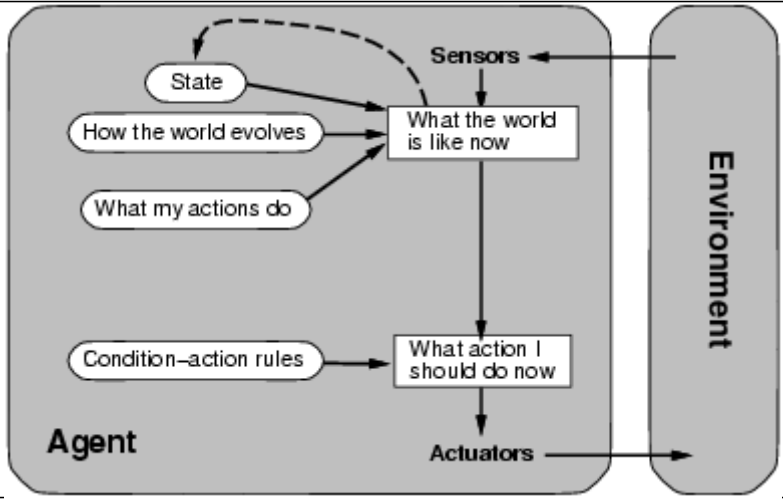

Figure 1.12 A model based reflex agent

function REFLEX-AGENT-WITH-STATE(*percept*) **returns** an action
  static: *rules*, a set of condition-action rules
  state, a description of the current world state
  action, the most recent action.
state ← UPDATE-STATE(*state*, *action*, *percept*)
rule ← RULE-MATCH(*state*, *rule*)
action ← RULE-ACTION[*rule*]
return *action*

Figure 1.13 Model based reflex agent. It keeps track of the current state of the world using an internal model. It then chooses an action in the same way as the reflex agent.

Goal-based agents

Knowing about the current state of the environment is not always enough to decide what to do. For example, at a road junction, the taxi can turn left, turn right, or go straight on. The correct decision depends on where the taxi is trying to get to. In other words, as well as a current state description, the agent needs some sort of **goal** information that describes situations that are desirable-for example, being at the passenger's destination. The agent program can combine this with information about the results of possible actions (the same information as was used to update internal state in the reflex agent) in order to choose actions that achieve the goal. Figure shows the goal-based agent's structure.

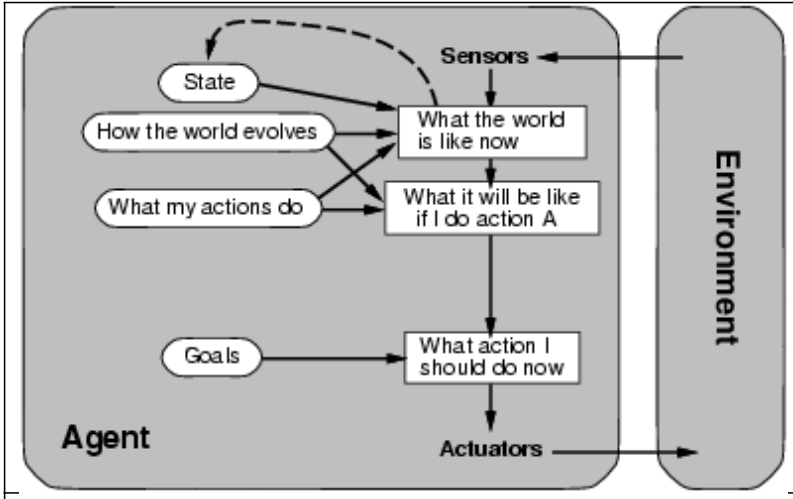

Figure 1.14 A goal based agent

Utility-based agents

Goals alone are not really enough to generate high-quality behavior in most environments. For example, there are many action sequences that will get the taxi to its destination (thereby achieving the goal) but some are quicker, safer, more reliable, or cheaper than others. Goals just provide a crude binary distinction between "happy" and "unhappy" states, whereas a more general **performance measure** should allow a comparison of different world states according to exactly how happy they would make the agent if they could be achieved. Because "happy" does not sound very scientific, the customary terminology is to say that if one world state is preferred to another, then it has higher **utility** for the agent.

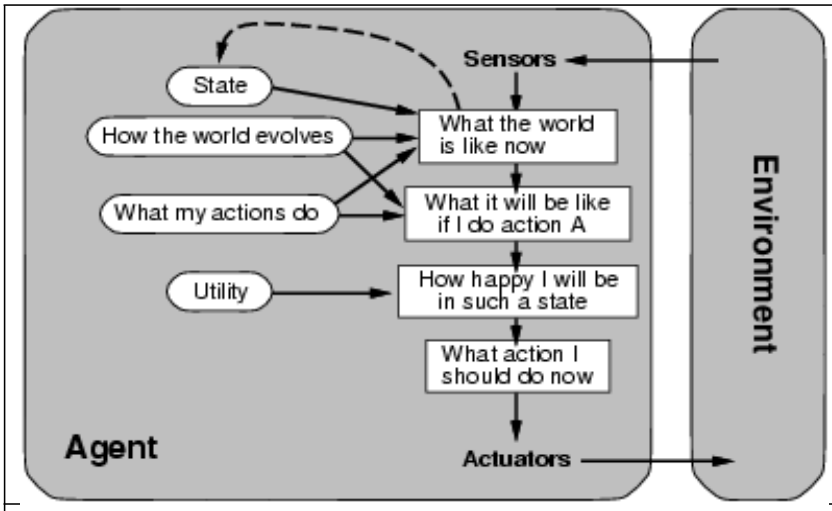

Figure 1.15 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

- Certain goals can be reached in different ways.
  - Some are better, have a higher utility.
- Utility function maps a (sequence of) state(s) onto a real number.
- Improves on goals:
  - Selecting between conflicting goals
  - Select appropriately between several goals based on likelihood of success.

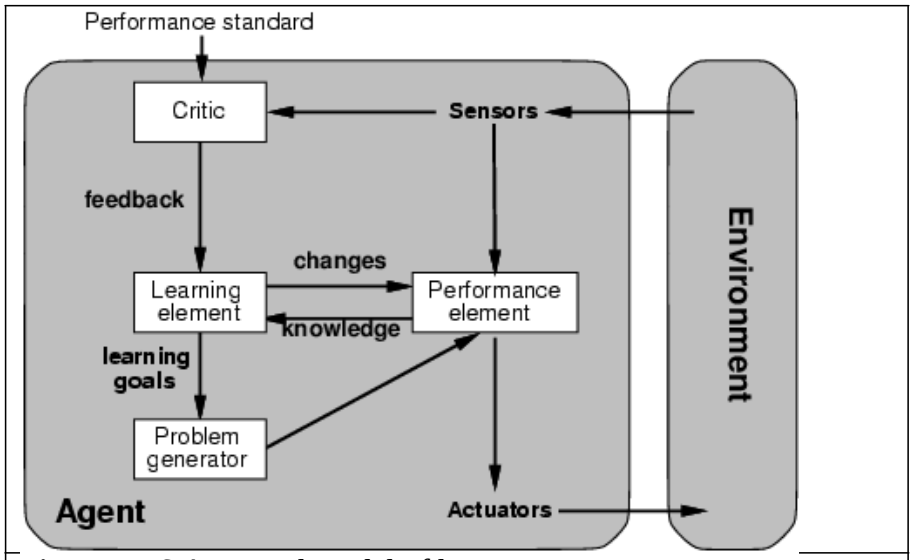

Figure 1.16 A general model of learning agents.

- All agents can improve their performance through **learning.**

A learning agent can be divided into four conceptual components, as shown in Figure 1.15 The most important distinction is between the **learning element**, which is responsible for making improvements, and the **performance element**, which is responsible for selecting external actions. The performance element is what we have previously considered to be the entire agent: it takes in percepts and decides on actions. The learning element uses feedback from the **critic** on how the agent is doing and determines how the performance element should be modified to do better in the future.

The last component of the learning agent is the **problem generator**. It is responsible for suggesting actions that will lead to new and **informative experiences**. But if the agent is willing to explore a little, it might discover much better actions for the long run. The problem generator's job is to suggest these **exploratory actions**. This is what scientists do when they carry out experiments.

Summary: Intelligent Agents

- An **agent** perceives and acts in an environment, has an architecture, and is implemented by an agent program.
- Task environment – **PEAS (P**erformance**, E**nvironment, **A**ctuators, **S**ensors**)**
- The most challenging environments are inaccessible, nondeterministic, dynamic, and continuous.
- An **ideal agent** always chooses the action which maximizes its expected performance, given its percept sequence so far.
- An **agent program** maps from percept to action and updates internal state.
  - **Reflex agents** respond immediately to percepts.

    - simple reflex agents
    - model-based reflex agents
  - **Goal-based agents** act in order to achieve their goal(s).
  - **Utility-based agents** maximize their own utility function.
- All agents can improve their performance through **learning.**