Session 2: Introduction to Agents

Course Title: Computational Intelligence

Course Code: CSC401A

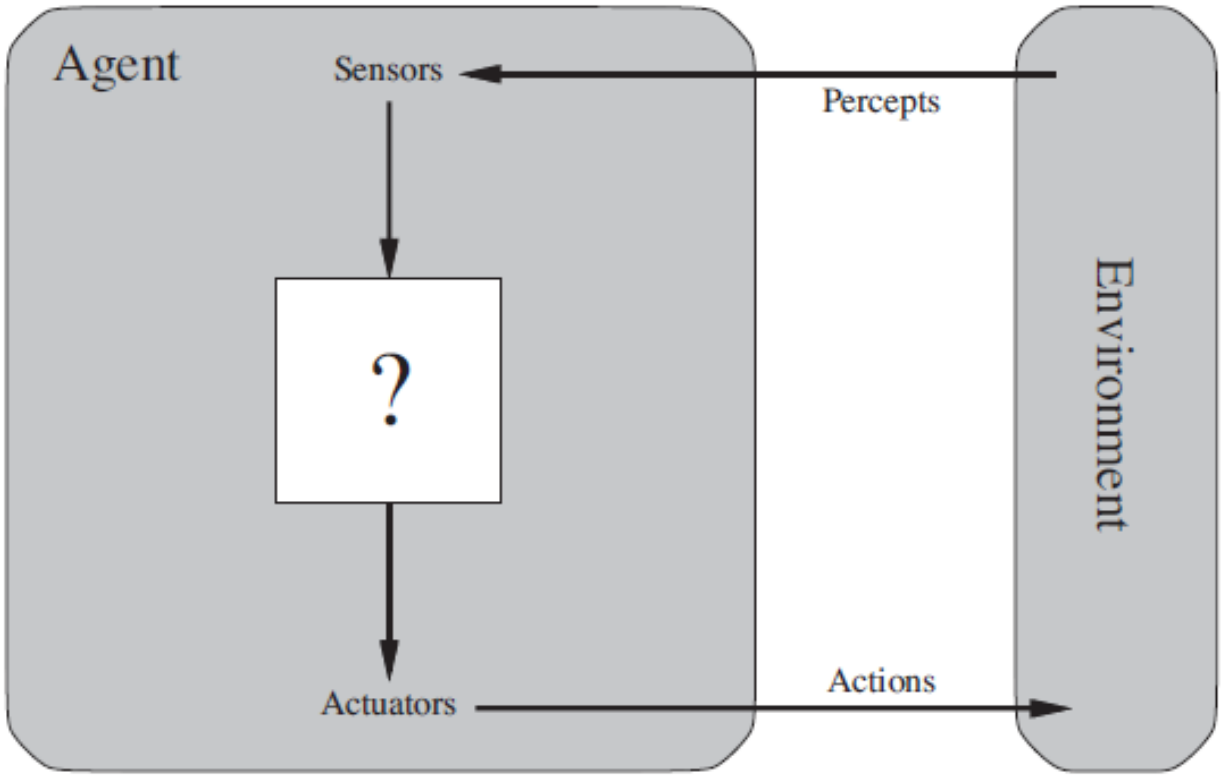
Vaishali R Kulkarni

Department of Computer Science and Engineering
Faculty of Engineering and Technology
M. S. Ramaiah University of Applied Sciences
Email: *vaishali.cs*.et@msruas.ac.in

Intelligent Agents

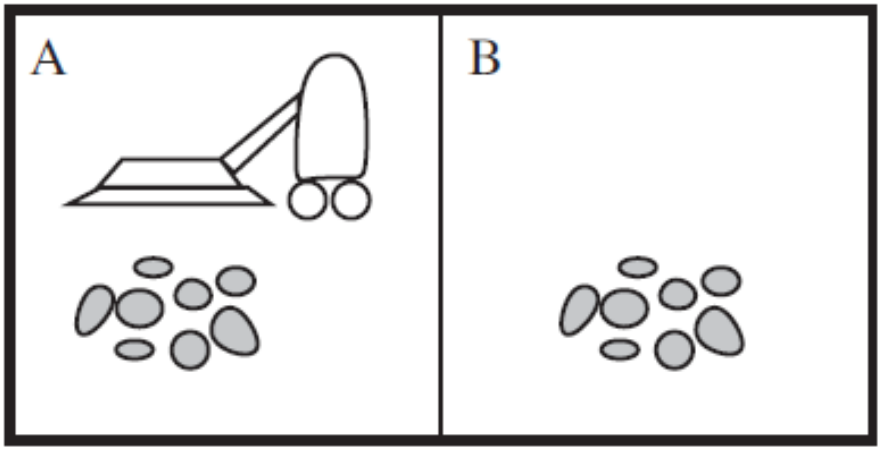
- An agent is just something that acts. Computer agents are expected to operate autonomously, perceive their environment and persist over a prolonged time period.
- A rational agent is one that acts so as to achieve the best outcome. Rational agent change, and create and pursue goals.
- An **intelligent agent** is anything that can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through **actuators**.
- Eg: A human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators
- A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators
- A software agent receives keystrokes, file contents, and network packets as sensory inputs and acts on the environment by displaying on the screen, writing files, and sending network packets.

Agents Interaction



Some Important Terms

- **Percept**: Refer to the agent's perceptual inputs at any given instant.
- **Percept sequence**: The complete history of everything the agent has ever perceived.
- **Agent function:** Mathematical description of agent's behavior
- **Agent program**: It is a concrete implementation, running within some physical system. Example of vacuum cleaner agent is as follows:
- There are just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck; otherwise, move to the other square



Tabulation of Vacuum Cleaner Agent

Percept sequence	Action
$[A, Clean]$	$Right$
$[A, Dirty]$	$Suck$
$[B, Clean]$	$Left$
$[B, Dirty]$	$Suck$
$[A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Dirty]$	$Suck$
⋮	⋮
$[A, Clean], [A, Clean], [A, Clean]$	$Right$
$[A, Clean], [A, Clean], [A, Dirty]$	$Suck$
⋮	⋮

Rational Agent

- Rational agent is an intelligent agent that should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and built-in knowledge.
- Consider a vacuum cleaner agent, it would be irrational under different circumstances. For example, once all the dirt is cleaned up, the agent will oscillate needlessly back and forth; This unnecessary movement will make the agent poor.
- A better agent for this case would do nothing once it is sure that all the squares are clean. If clean squares can become dirty again, the agent should occasionally check and re-clean them if needed.

PEAS

- Agents can be grouped under the heading of the task environment defined in PEAS (Performance, Environment, Actuators, Sensors) description.
- PEAS description of the task environment for an automated taxi.

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard

Agent types and their PEAS descriptions

Agent Type	Performance Measure	Environment	Actuators	Sensors
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Environment in Agents

- The range of task environments in AI is vast. Environments are classified as follows:
- **Fully observable** vs. **partially observable**: If an agent's sensors give it access to the complete state of the environment at each point in time, the task environment is fully observable.
- **Single agent** vs. **multiagent**: For example, an agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is in a two agent environment. For e.g.. chess is a **competitive** multiagent environment. In the taxi-driving environment, is a partially **cooperative** and **competitive**. The agent-design problems in multiagent environments are often quite different from those in single-agent environments; for e.g.., **communication** often emerges as a rational behavior in multiagent environments; in some competitive environments, **randomized behavior** is rational because it avoids the pitfalls of predictability.

Environments contd.

- **Deterministic** vs. **stochastic:** If the next state of the environment is completely determined by the current state and the action executed by the agent, then the environment is deterministic; otherwise, it is stochastic.
- **Stochastic** implies that uncertainty about outcomes is quantified in terms of probabilities.
- **Nondeterministic** environment is one in which actions are characterized by their *possible* outcomes, but no probabilities are attached to them. Nondeterministic environment descriptions are usually associated with performance measures that require the agent to succeed for *all possible* outcomes of its actions

Environments cont.

- **Episodic vs. sequential**: In an episodic task environment, the agent's experience is divided into atomic episodes. In each episode the agent receives a percept and then performs a single action. Crucially, the next episode does not depend on the actions taken in previous episodes.
- **For e.g**. , an agent that has to spot defective parts on an assembly line bases each decision on the current part, regardless of previous decisions; moreover, the current decision doesn't affect whether the next part is defective.
- In **sequential environments**, on the other hand, the current decision could affect all future decisions. Chess and taxi driving are sequential: in both cases, short-term actions can have long-term consequences. Episodic environments are much simpler than sequential environments because the agent does not need to think ahead

Environments cont.

- **Static vs. dynamic:** If the environment can change while an agent is deliberating, then the environment is dynamic for that agent; otherwise, it is static.
- **Static** environments are easy to deal with because the agent need not keep looking at the world while it is deciding on an action, nor need it worry about the passage of time. Dynamic environments, on the other hand, are continuously asking the agent what it wants to do; if it hasn't decided yet, that counts as deciding to do nothing.
- If the environment itself does not change with the passage of time but the agent's performance score does, then the environment is **semi-dynamic**. Taxi driving is **dynamic**. Chess, when played with a clock, is **semi-dynamic**. Crossword puzzles are **static**.

Environments cont.

- **Discrete vs. continuous**: The discrete/continuous distinction applies to the state of the environment, to the way time is handled, and to the percepts and actions of the agent. For e.g. the chess environment has a finite number of distinct states (excluding the clock).

- Chess also has a discrete set of percepts and actions. Taxi driving is a continuous-state and continuous-time problem: the speed and location of the taxi and of the other vehicles sweep through a range of continuous values and do so smoothly over time. Taxi-driving actions are also continuous (steering angles, etc.). Input from digital cameras is discrete, strictly speaking, but is typically treated as representing continuously varying intensities and locations.

Environments cont.

- **Known vs. unknown**: In a known environment, the outcomes for all actions are given. If the environment is unknown, the agent will have to learn how it works in order to make good decisions.
- The distinction between known and unknown environments is not the same as the one between fully and partially observable environments. It is quite possible for a known environment to be partially observable—for example, in solitaire card games, I know the rules but am still unable to see the cards that have not yet been turned over. Conversely, an unknown environment can be fully observable—in a new video game, the screen may show the entire game state but I still don't know what the buttons do until I try them.

Examples of task environments and their characteristics

Task Environment	Observable	Agents	Deterministic	Episodic	Static	Discrete
Crossword puzzle	Fully	Single	Deterministic	Sequential	Static	Discrete
Chess with a clock	Fully	Multi	Deterministic	Sequential	Semi	Discrete
Poker	Partially	Multi	Stochastic	Sequential	Static	Discrete
Backgammon	Fully	Multi	Stochastic	Sequential	Static	Discrete
Taxi driving	Partially	Multi	Stochastic	Sequential	Dynamic	Continuous
Medical diagnosis	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Image analysis	Fully	Single	Deterministic	Episodic	Semi	Continuous
Part-picking robot	Partially	Single	Stochastic	Episodic	Dynamic	Continuous
Refinery controller	Partially	Single	Stochastic	Sequential	Dynamic	Continuous
Interactive English tutor	Partially	Multi	Stochastic	Sequential	Dynamic	Discrete

Structure of Agents

AGENT = ARCHITECTURE + PROGRAM

- The agent programs take the current percept as input from the sensors and return an action to the actuators. If the agent's actions need to depend on the entire percept sequence, the agent remembers the percepts. Example of table driven agent is as follows:

```
function TABLE-DRIVEN-AGENT(percept) returns an action
  persistent: percepts, a sequence, initially empty
              table, a table of actions, indexed by percept sequences, initially fully specified

  append percept to the end of percepts
  action ← LOOKUP(percepts, table)
  return action
```

Figure 2.7 The TABLE-DRIVEN-AGENT program is invoked for each new percept and returns an action each time. It retains the complete percept sequence in memory.

Basic kinds of agent programs

- Simple reflex agents: It does not bother about previous state
- Model-based reflex agents
- Goal-based agents
- Utility-based agents
- **Simple reflex agents** respond directly to percepts, whereas **model-based reflex agents** maintain internal state to track aspects of the world that are not evident in the current percept. **Goal-based agents** act to achieve their goals, and **utility-based agents** try to maximize their own expected *happiness*

Simple Reflex Agents

- Simple reflex agents: It does not bother about previous state
- Works on current state
- Base on condition – action rule
- If condition = true then ACTION
- Else
- NO ACTION

Drawback: Limited intelligence, no knowledge about non-perceptual parts, working in partial environment, ends up in infinite loops. Eg: Self driving cars, humans, robots

Reflex Agent

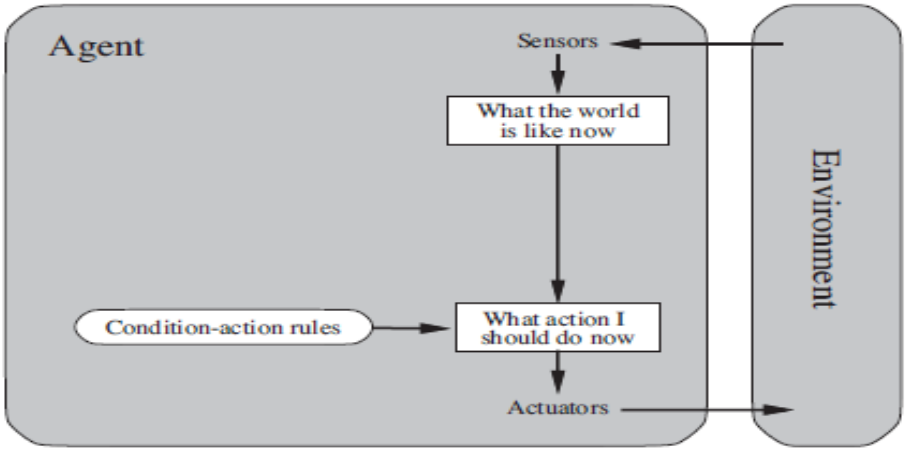


Figure 2.9 Schematic diagram of a simple reflex agent.

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

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

Figure 2.10 A simple reflex agent. It acts according to a rule whose condition matches the current state, as defined by the percept.

If the car in front brakes and its brake lights come on, then user initiate braking. Rule for this example is: *if car-in-front-is-braking then initiate-braking*.

Model-based reflex agents

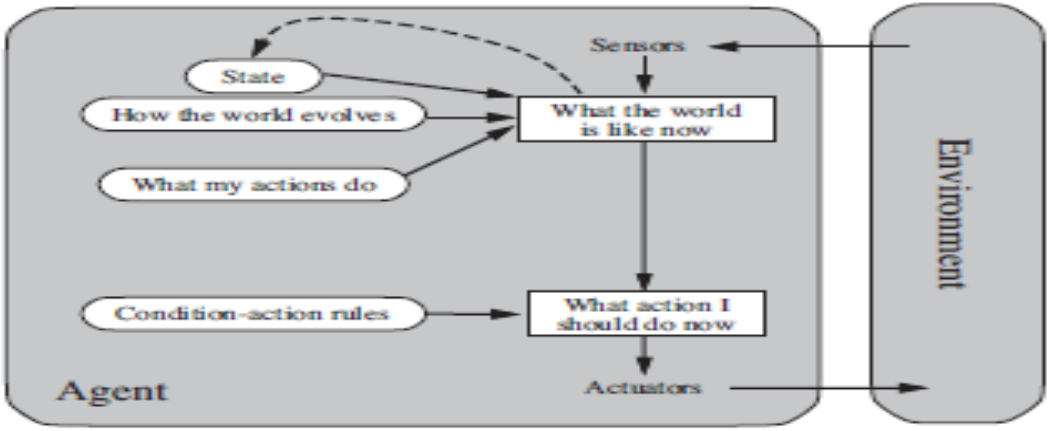


Figure 2.11 A model-based reflex agent.

```
function MODEL-BASED-REFLEX-AGENT(percept) returns an action
  persistent: state, the agent's current conception of the world state
              model, a description of how the next state depends on current state and action
              rules, a set of condition–action rules
              action, the most recent action, initially none

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

Figure 2.12 A 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.

The taxi may be driving back home, and it may have a rule telling it to fill up with gas on the way home unless it has at least half a tank. Although "driving back home" may seem taxi's destination but there may be internal state.

Model Based

- It works by finding a rule whose condition matches the current situation
- It works on partial environment
- It finds a rule with that it matches the environment

Goal-based agents

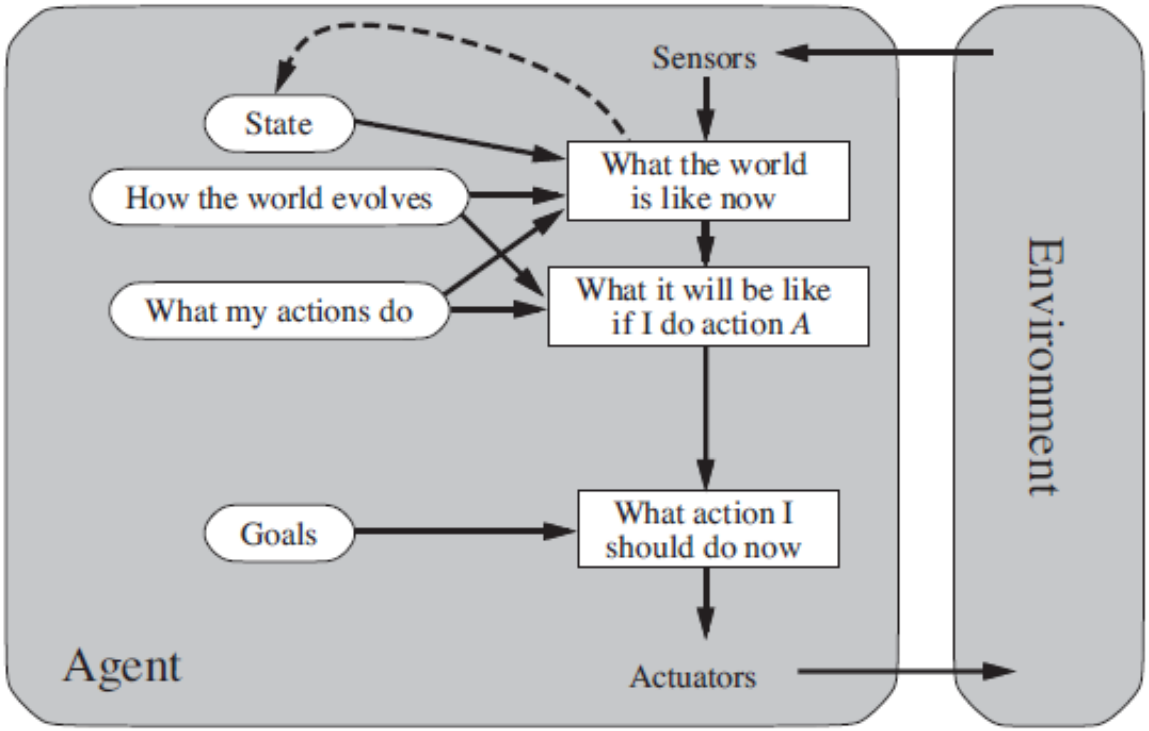


Figure 2.13 A model-based, goal-based agent. It keeps track of the world state as well as a set of goals it is trying to achieve, and chooses an action that will (eventually) lead to the achievement of its goals.

Google's Waymo driverless cars are good examples of a goal-based agent when they are programmed with an end destination, or goal, in mind. The car will then ''think'' and make the right decisions in order to deliver the passenger where they intended to go.

Model Based

- It works by finding a rule whose condition matches the current situation
- It works on partial environment
- It finds a rule with that it matches the environment

Utility-based agents

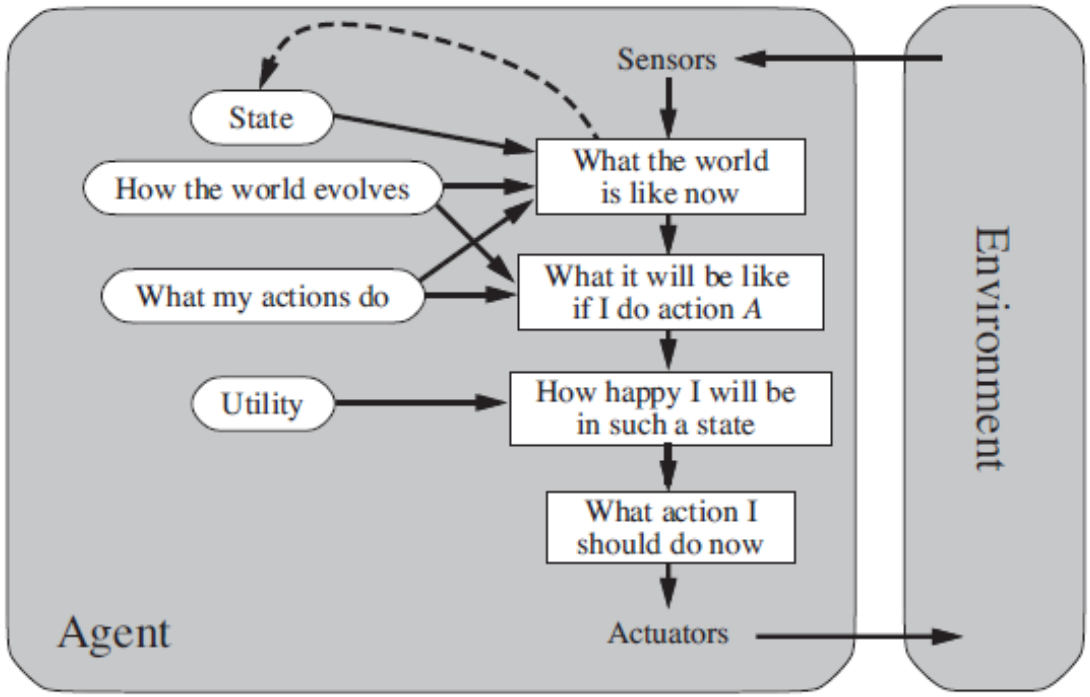


Figure 2.14 A model-based, utility-based agent. It uses a model of the world, along with a utility function that measures its preferences among states of the world. Then it chooses the action that leads to the best expected utility, where expected utility is computed by averaging over all possible outcome states, weighted by the probability of the outcome.

Home thermostat that knows to start heating or cooling your house based on reaching a certain temperature.

Learning agents

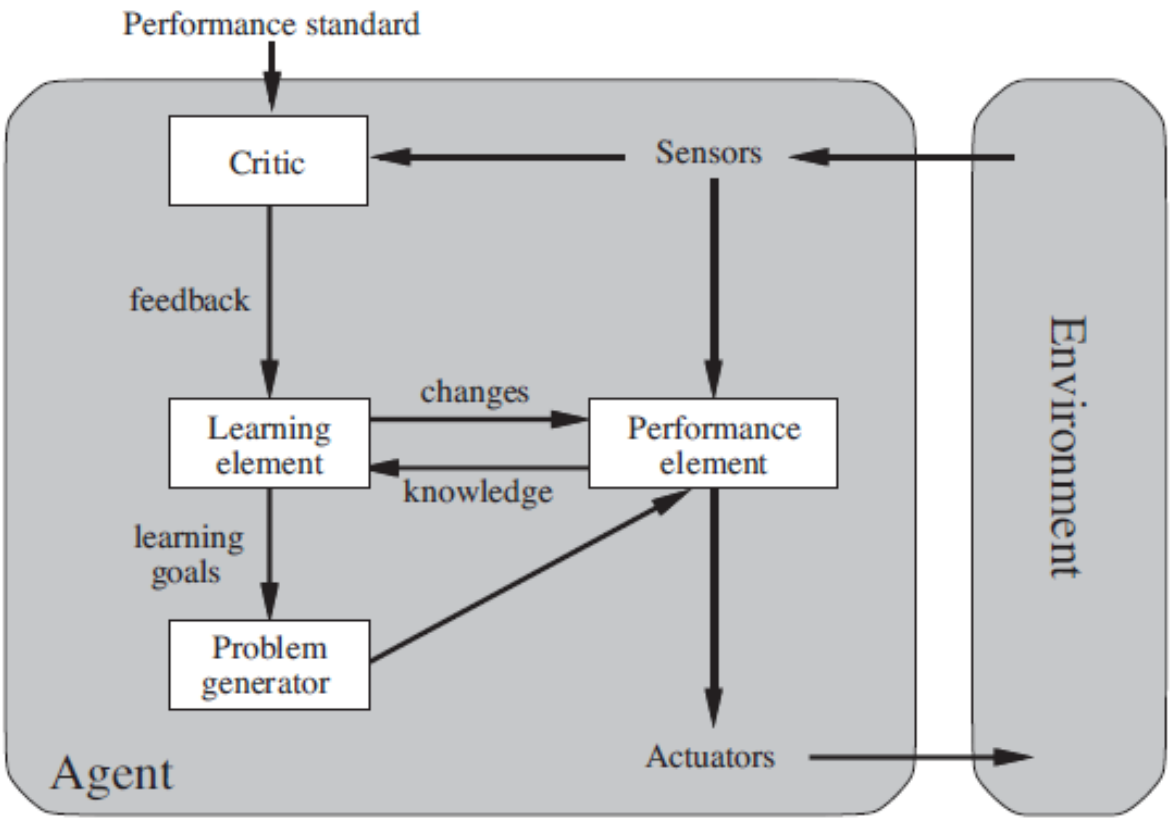


Figure 2.15 A general learning agent.

Some **intelligent** software **agents** are able to learn and adapt with Machine Learning. Microsoft's Cortana, Apple's **Siri**, Google's Google Now, Braina, Amazon Echo (and more) are designed to learn and act without supervision

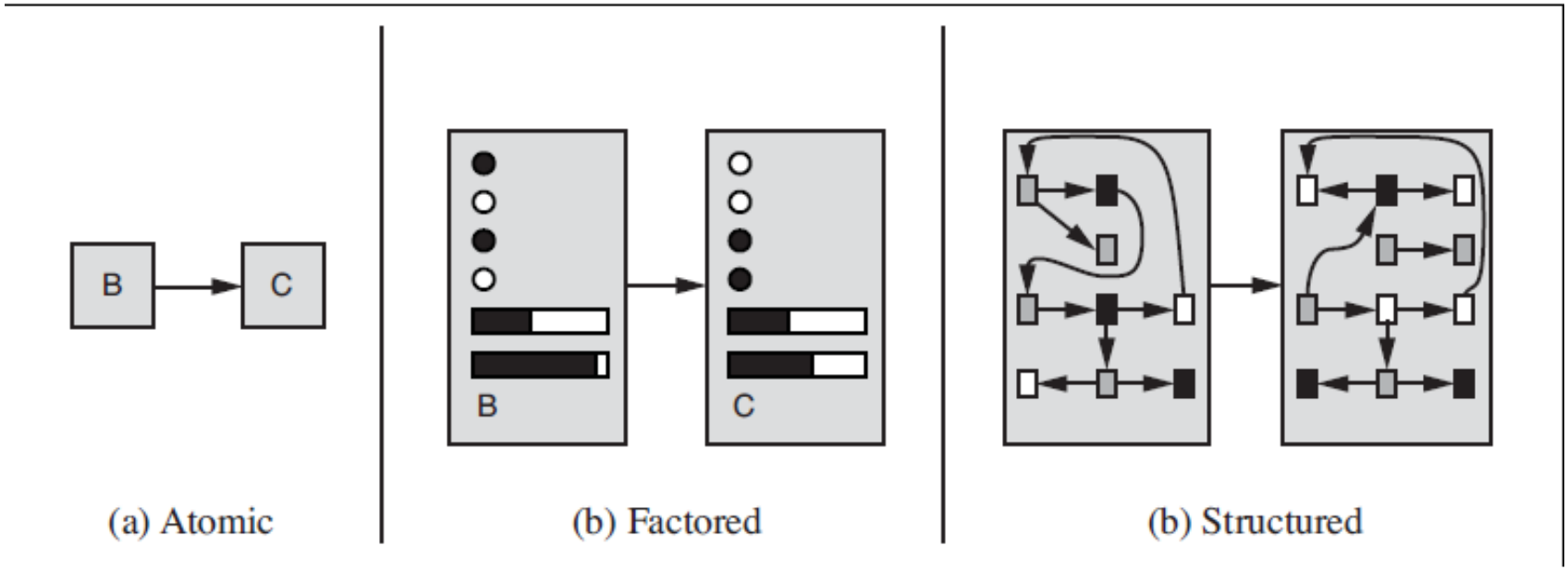
Learning agents

A learning agent can be divided into four conceptual components.

- **The critic:** It tells the learning element how well the agent is doing with respect to a fixed performance standard. The critic is necessary because the percepts themselves provide no indication of the agent's success. For example, a chess program could receive a percept indicating that it has checkmated its opponent, but it needs a performance standard to know that this is a good thing; the percept itself does not say so.
- **The learning element**: Responsible for making improvements. 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 performance element:** Responsible for selecting external actions. It takes in percepts and decides on actions.
- **Problem Generator:** It is responsible for suggesting actions that will lead to new and informative experiences. The problem generator's job is to suggest more exploratory actions to performance element.

Working of Agent Components

- Three ways to represent states and the transitions between agent components.
- Atomic representation: a state (such as B or C) is a black box with no internal structure
- Factored representation: a state consists of a vector of attribute values; values can be Boolean, real valued, or one of a fixed set of symbols.
- Structured representation: a state includes objects, each of which may have attributes of its own as well as relationships to other objects.



Atomic representation

- In an atomic representation each state of the world is indivisible—it has no internal structure. Consider the problem of finding a driving route from one end of a country to the other via some sequence of cities. For the purposes of solving this problem, it may suffice to reduce the state of world to just the name of the city we are in—a single atom of knowledge. It can be also referred as “black box” whose only discernible property is that of being identical to or different from another black box. Algorithms in atomic representation are:
- **Search** and **game-playing, Hidden Markov models ,** and **Markov decision processes**

Factored representation

- Consider a higher-fidelity description for the same problem, where we need to be concerned with more than just location in one city or another; there may be a need to pay attention to how much gas is in the tank, current GPS coordinates, whether or not the oil warning light is working, how much spare change we have for toll crossings, what station is on the radio, etc..
- A factored representation splits up each state into a fixed set of variables or attributes, each of which can have a value. While two different atomic states have nothing in common—they are just different black boxes—two different factored states can share some attributes (such as being at some particular GPS location) and not others (such as having lots of gas or having no gas); this makes it much easier to work out how to turn one state into another.
- With factored representations uncertainty can be expressed. Many **important areas** of AI are based on factored representations, such as **constraint satisfaction algorithms, propositional logic, planning and Bayesian networks**

Structured representation

- For many purposes, we need to understand the world as having things in it that are related to each other, not just variables with values.
- For example, we might notice that a large truck ahead of us is reversing into the driveway of a dairy farm but a cow has got loose and is blocking the truck's path. A factored representation for this problem is: ***Truck_Ahead_Backing_Into_Dairy_Farm_Driveway_Blocked_By_Loose_Cow with value true or false***.
- Instead, a structured representation, in which objects such as cows and trucks and their various and varying relationships can be described explicitly.
- Structured representations underlie relational databases and **first-order logic, first-order probability models, knowledge-based learning and much of natural language understanding**. Almost everything that humans express in natural language concerns objects and their relationships.

References

- Artificial Intelligence A Modern Approach Third Edition by Stuart J. Russell and Peter Norvig