Logic for Multiagent Systems (Supplementary material)

Master 1st Year, 1st Semester 2024/2025

Laurențiu Leuștean

Web page: https://cs.unibuc.ro//~lleustean/Teaching/2024-LMS/index.html

Agents

Textbook:

Michael Wooldridge, An Introduction to MultiAgent Systems, Second Edition, John Wiley & Sons, 2009

We also use

Lecture slides/handouts, made available by Michael Wooldridge here

Agents

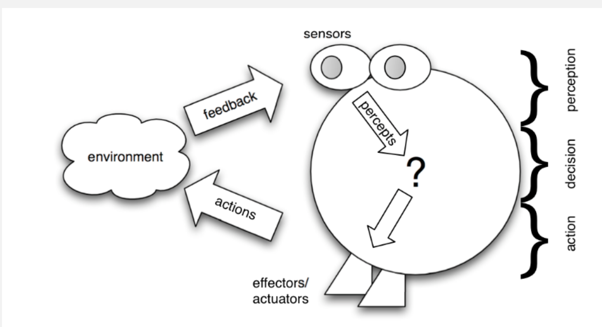


Figure 1: An agent in its environment

- Figure 1 gives an abstract view of an agent in its environment
- The agent takes sensory input from the environment, and produces, as output, actions that affect it. The interaction is usually an ongoing, non-terminating one.

- Usually, an agent will not have complete control over its environment.
- It will have at best partial control, in that it can influence it.
- From the point of view of the agent, this means that the same action performed twice in apparently identical circumstances might appear to have entirely different effects, and in particular, it may fail to have the desired effect.
- Thus agents in all but the most trivial of environments must be prepared for the possibility of failure.

Agents

Make formal the abstract view of agents.

- Assume the environment may be in any of a finite set E of discrete, instantaneous states:

$$E = \{e', e'', \dots\}$$

- An agent is assumed to have a repertoire of possible actions available:

$$Ac = \{\alpha', \alpha'', \dots\}$$

- Actions transform the state of the environment.
- We assume that the set Ac of actions contains a special action *null*, with the meaning that nothing will be done.
- States are denoted also by e_0, e_1, ...
- Actions are denotes also by α_0, α_1, ...

Agents

The basic model of agents interacting with their environments is as follows:

- The environment starts in some state.
- The agent begins by choosing an action to perform on that state.
- As a result of this action, the environment can respond with a number of possible states. However, only one state will actually result — though, of course, the agent does not know in advance which it will be.
- On the basis of this second state, the agent again chooses an action to perform.
- The environment responds with one of a set of possible states.
- The agent then chooses another action; and so on.

Agents

A run over E and Ac is a finite sequence of interleaved environment states and actions.

Definition 0.1

A run r over E and Ac is a finite sequence

$$r = (x_0, x_1, x_2, \dots, x_n),$$

where $n \in \mathbb{N}$ and for all $k \in \mathbb{N}$, $x_{2k} \in E$ and $x_{2k+1} \in Ac$.

Runs are denoted by $r, r', \dots$. We write a run r as follows:

$$r: \quad e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{u-2}} e_{u-1} \xrightarrow{\alpha_{u-1}} e_u$$

or

$$r: \quad e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{u-2}} e_{u-1} \xrightarrow{\alpha_{u-1}}$$

Agents

Notation 0.2

- $\mathcal{R}$ *denotes the set of all runs (over E and Ac).*
- $\mathcal{R}^{Ac}$ *is the subset of these that end with an action.*
- $\mathcal{R}^{E}$ *is the subset of these that end with an environment state.*

Definition 0.3

A function $\tau : \mathcal{R}^{Ac} \to 2^E$ is said to be a state transformer function.

- A state transformer function maps a run $r \in \mathcal{R}^{Ac}$ to a set $\tau(r)$ of possible environment states that could result from performing the action.
- State transformer functions represent the effect that an agent's actions have on an environment.

If $\tau(r) = \emptyset$, then there are no possible successor states to r. In this case, we say that the run r has ended or that r is a terminated run.

Recall: For any set A, 2^A is the set of all subsets of A:

$$2^A = \{B \mid B \subseteq A\}.$$

Definition 0.4

An environment is a triple $Env = (E, e_0, \tau)$, where E is the set of environment states, $e_0 \in E$ is an initial state, and τ is a state transformer function.

Environments are:

- history dependent. The next state of an environment is not solely determined by the action performed by the agent and the current state of the environment. The actions made earlier by the agent also play a part in determining the current state.
- non-deterministic. There is uncertainty about the result of performing an action in some state.

We introduce a model of the agents that inhabit systems.

Definition 0.5

An agent is a function $Ag : \mathcal{R}^E \to Ac$ mapping runs (assumed to end with an environment state) to actions.

- An agent makes a decision about what action to perform based on the history of the system.
- Agents are deterministic.

Definition 0.6

A system is a pair (Ag, Env) containing an agent Ag and an environment $Env = (E, e_0, \tau)$.

Definition 0.7

A run in the system (Ag, Env) is a run $r = (x_0, x_1, x_2, \ldots, x_n)$ over E and Ac satisfying the following:

- $x_0 = e_0$.
- *for all* $k \geq 0$:

$$x_{2k+1} = Ag(x_0, \ldots, x_{2k}) \text{ and } x_{2k+2} \in \tau(x_0, \ldots, x_{2k+1}).$$

- *r is terminated in the following sense:*
  - *if $x_n \in E$, then $Ag(r) = null$;*
  - *if $x_n \in Ac$, then $\tau(r) = \emptyset$.*

We also say that r is a run of the agent Ag in the environment Env.

Notation 0.8

We denote by $\mathcal{R}(Ag, Env)$ the set of all runs in the system (Ag, Env).

Agents

Let $r \in \mathcal{R}^{Ac}$,

$$r = e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{u-2}} e_{u-1} \xrightarrow{\alpha_{u-1}}$$

Then $r \in \mathcal{R}(Ag, Env)$ iff the following are satisfied:

- $\alpha_0 = Ag(e_0)$.
- For all $j = 1, \dots, u-1$,

$$\begin{aligned} e_j &\in \tau(e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{j-2}} e_{j-1} \xrightarrow{\alpha_{j-1}}) \\ \alpha_j &= Ag(e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{j-1}} e_j). \end{aligned}$$

- $\tau(r) = \emptyset$.

Agents

Let $r \in \mathcal{R}^{E}$,

$$r = e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{u-2}} e_{u-1} \xrightarrow{\alpha_{u-1}} e_u$$

Then $r \in \mathcal{R}(Ag, Env)$ iff the following are satisfied:

- $\alpha_0 = Ag(e_0)$.
- For all $j = 1, \dots, u$,

$$\begin{aligned} e_j &\in \tau(e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{j-2}} e_{j-1} \xrightarrow{\alpha_{j-1}}) \\ \alpha_{j-1} &= Ag(e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_{j-2}} e_{j-1}). \end{aligned}$$

- $Ag(r) = null$.

Agents

Definition 0.9

Two agents Ag_1 and Ag_2 are said to be

- *behaviourally equivalent with respect to environment Env if and only if $\mathcal{R}(Ag_1, Env) = \mathcal{R}(Ag_2, Env)$.*
- *behaviourally equivalent if they are behaviourally equivalent with respect to all environments.*

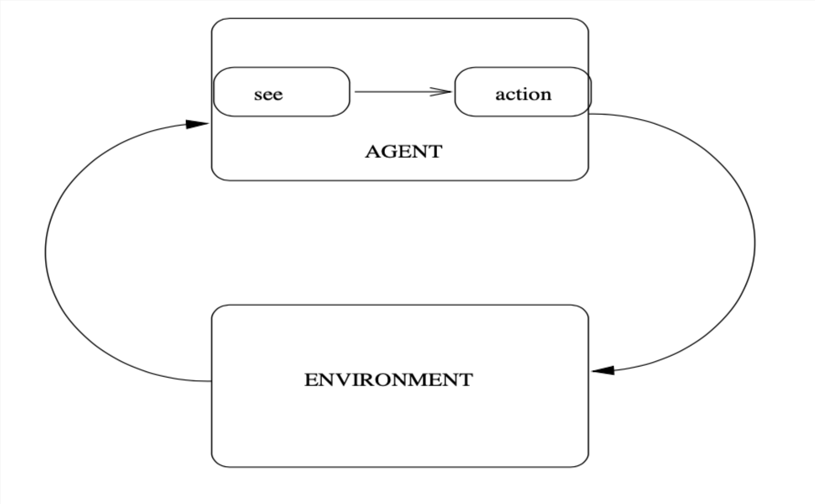
Agents

This view of agents is too abstract. It does not help us to construct them, since it gives us no clues about how to design the decision function action.

- We refine our abstract model of agents, by breaking it down into subsystems.
- We make design choices on these subsystems — what data and control structures will be present.
- An agent architecture is essentially a map of the internals of an agent — its data structures, the operations that may be performed on these data structures, and the control flow between these data structures.
- There are different types of agent architectures, with very different views on the data structures and algorithms that will be present within an agent.

Perception

One high-level design decision is the separation of an agent's decision function into perception and action subsystems.



Perception

- The perception function *see* captures the agent's ability to observe its environment. Example: a video camera or an infra-red sensor on a mobile robot.
- The output of the see function is a percept — a perceptual input.
- The action function represents the agent's decision making process.

Let Per be a nonempty set of percepts.

Definition 0.10

The see and action functions are defined as follows:

$$see : E \to Per \qquad \text{and} \qquad action : Per^* \to Ac.$$

Recall: For any set A, A^* is the set of all finite sequences of elements of A:

$$A^* = \{a_1 a_2 \dots a_n \mid n \in \mathbb{N} \text{ and } a_i \in A \text{ for all } i = 1, \dots, n\}.$$

Perception

These simple definitions allow us to explore some interesting properties of agents and perception.

Suppose that we have two environment states $e_1, e_2 \in E$ such that $e_1 \neq e_2$, but $see(e_1) = see(e_2)$. Then e_1 and e_2 are mapped to the same percept, and the agent receives the same perceptual information from each of them. As far as the agent is concerned, e_1 and e_2 are indistinguishable.

Definition 0.11

The relation $\equiv$ on E is defined as follows: for every $e_1, e_2 \in E$,

$$e_1 \equiv e_2 \quad \text{iff} \quad see(e_1) = see(e_2).$$

Remark 0.12

$\equiv$ is an equivalence relation on E.

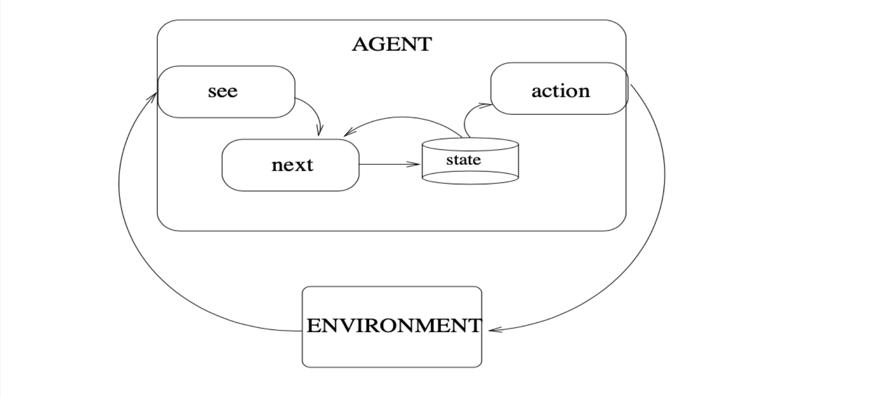
Perception

- $\equiv$ partitions E into mutually indistinguishable sets of states, namely the different equivalence classes $[e]$, were $e \in E$.

- If $[e] = \{e\}$ for every $e \in E$, then $see(e_1) \neq see(e_2)$ for every states $e_1 \neq e_2$. The agent can distinguish every state — the agent has perfect perception in the environment.
- If $[e] = E$ for every $e \in E$, then $see(e_1) = see(e_2)$ for every states e_1, e_2. The agent's perceptual ability is non-existent, it cannot distinguish between any different states. As far as the agent is concerned, all environment states are identical.

Agents with state

We now consider agents that maintain state.



These agents have some internal data structure, which is typically used to record information about the environment state and history.

Agents with state

Let I be the set of all internal states of the agent.

Definition 0.13

The see and action functions are defined as follows:

$$see : E \to Per \qquad \text{and} \qquad action : I \to Ac.$$

The perception function *see* is unchanged. The action-selection function *action* takes as inputs internal states.

Definition 0.14

The function next is defined as follows:

$$next : I \times Per \to I.$$

Agents with state

The behaviour of a state-based agent:

- The agent starts in some initial internal state i_0.
- It then observes its environment state e, and generates a percept $see(e)$.
- The internal state of the agent is then updated to $i_1 := next(i_0, see(e))$.
- The action selected by the agent is $\alpha := action(i_1)$.
- The agents performs action α.
- The agent enters another cycle: perceives the world via *see*, updates its state via *next*, and chooses an action to perform via *action*.

Tasks for agents

- We build agents in order to carry out tasks for us.
- The tasks to be carried out must be specified by us in some way
- How to specify these tasks? How to tell the agent what to do?

One way to to do this: write a program for the agent to execute.

- Advantage: no uncertainty about what the agent will do; it will do exactly what we told it to, and no more.
- Disadvantage: we have to think about exactly how the task will be carried out ourselves; if unforeseen circumstances arise, the agent executing the task will be unable to respond accordingly.

Tasks for agents

- We want to tell our agent what to do without telling it how to do it.
- One way of doing this is to define tasks indirectly, via some kind of performance measure.
- One possibility: associate utilities with states of the environment.

- A utility is a numeric value representing how 'good' a state is: the higher the utility, the better.
- The task of the agent is then to bring about states that maximize utility.
- We do not specify to the agent how this is to be done.

Utility functions

Definition 0.15

A utility function (or task specification) is a function $u : E \to \mathbb{R}$.

What is the overall utility of a run?

- minimum utility of a state on run?
- maximum utility of a state on run?
- sum of utilities of all states on run?
- average utility of all states on run?

Main disadvantage:

- assigns utilities to local states.
- difficult to specify a long-term view when assigning utilities to individual states.

Utility functions

Idea: assign a utility not to individual states, but to runs.

Definition 0.16

A utility function (or task specification) is a function $u : \mathcal{R} \to \mathbb{R}$.

If we are concerned with agents that must operate independently over long periods of time, then this approach is appropriate.

- The utility-based approach works well in certain scenarios.
- Problems:
  - Sometimes it is difficult to define a utility function.
  - People don't think in terms of utilities. It is hard for people to specify tasks in these terms.

Tileworld

Tileworld was proposed as an experimental environment for evaluating agent architectures in
Martha E. Pollock, Marc Ringuette, Introducing the Tileworld: Experimentally Evaluating Agent Architectures, AAAI-90 Proceedings, 1990

- Simulated two dimensional grid environment on which there are agents, tiles, obstacles, and holes.
- An agent can move in four directions, up, down, left, or right, and if it is located next to a tile, it can push it.
- An obstacle is a group of immovable grid cells.
- Holes have to be filled up with tiles by the agent.
- An agent scores points by filling holes with tiles, the aim being to fill as many holes as possible.

Tileworld

- Holes appear randomly and exist for as long as their life expectancy, unless they disappear because of the agent's actions. The interval between the appearance of successive holes is called the hole gestation time.
- Tileworld is an example of a dynamic environment: starting in some randomly generated world state, based on parameters set by the experimenter, it changes over time in discrete steps, with the random appearance and disappearance of holes.
- The performance of an agent in the Tileworld is measured by running the Tileworld testbed for a predetermined number of time steps, and measuring the number of holes that the agent succeeds in filling.
- Experimental error is eliminated by running the agent in the environment a number of times, and computing the average of the performance.

Tileworld

Definition 0.17

The utility function is defined as follows:

$$u : \mathcal{R} \to \mathbb{R}, \qquad u(r) = \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r}$$

- u is normalized: $u(r) \in [0, 1]$ for every run r
- $u(r) = 1$: agent filled every hole that appeared in r
- $u(r) = 0$: agent did not fill any hole that appeared in r

Tileworld

- Despite its simplicity, Tileworld allows us to examine several important capabilities of agents.
- Examples of abilities of agents:
  - to react to changes in the environment
  - to exploit opportunities when they arise.

Tileworld

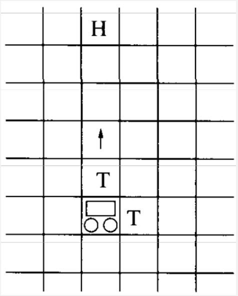


Figure 2: Tileworld example

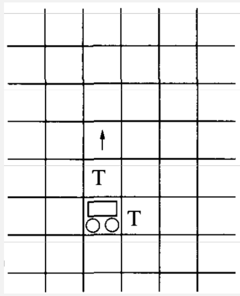


Figure 3: Tileworld example

Example 0.18

Suppose an agent is pushing a tile to a hole (Figure 2), when this hole disappears (Figure 3).
The agent should recognize this change in the environment, and modify its behaviour appropriately.

Tileworld

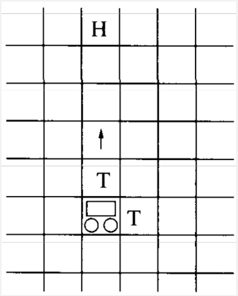


Figure 4: Tileworld example

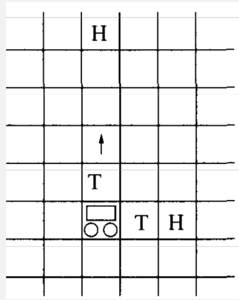


Figure 5: Tileworld example

Example 0.19

Suppose an agent is pushing a tile to a hole (Figure 4), when a hole appears to the right of the agent (Figure 5).
It would do better to push the tile to the right, than to continue to head north, for the simple reason that it only has to push the tile one step, rather than three.

Example 0.20

Expected utility

Let us denote $P(r \mid Ag, Env)$ the probability that run r occurs when agent Ag is placed in environment Env.
Obviously, $\sum_{r \in \mathcal{R}(Ag,Env)} P(r \mid Ag, Env) = 1.$

Definition 0.21

The expected utility of agent Ag in environment Env (given P, u) is defined as follows:

$$EU(Ag, Env) = \sum_{r \in \mathcal{R}(Ag,Env)} u(r)P(r \mid Ag, Env).$$

Optimal agents

Notation 0.22

Let AG denote the finite set of all agents acting in some environment.

Definition 0.23

An optimal agent in an environment Env is an agent Ag_{opt} that maximizes the expected utility:

$$Ag_{opt} = arg \max_{Ag \in AG} EU(Ag, Env).$$

- The fact that an agent is optimal does not mean that it will be best; only that on average, we can expect it to do best.
- The definition does not not give us any clues about how to implement this agent.
- There are agents that cannot be implemented on a real computer.

Bounded optimal agents

Suppose m is a particular computer.

Notation 0.24

AG_m denotes the set of agents that can be implemented on m:

$$AG_m = \{Ag \mid Ag \in AG \text{ and } Ag \text{ can be implemented on } m\}.$$

Definition 0.25

A bounded optimal agent in an environment Env, with respect to m, is an agent $Ag_{bopt} \in AG_m$ that maximizes the expected utility:

$$Ag_{bopt} = arg \max_{Ag \in AG_m} EU(Ag, Env).$$

- We consider only the agents that can actually be implemented on the machine that we have for the task.

Predicate task specifications

- A predicate task specification is one where the utility function acts as a predicate over runs.
- A utility function $u : \mathcal{R} \to \mathbb{R}$ is a predicate if the range of u is the set $\{0, 1\}$, that is if u assigns a run either 1 (true) or 0 (false).
- If $u(r) = 1$, we say that the run r satisfies the specification; the agent succeeds on the run r.
- If $u(r) = 0$, we say that the run r fails to satisfy the specification; the agent fails on the run r.

Definition 0.26

A predicate task specification is a mapping $\Psi : \mathcal{R} \to \{0, 1\}$.

Task environment

Definition 0.27

A task environment is a pair (Env, Ψ)*, where* Env *is an environment, and* Ψ *is a predicate task specification.*

Notation 0.28

TE *denotes the set of all task environments.*

A task environment specifies:

- the properties of the system the agent will inhabit (i.e. the environment Env);
- the criteria by which an agent will be judged to have either failed or succeeded (i.e. the specification Ψ).

Task environment

Notation 0.29

$\mathcal{R}_\Psi(Ag, Env)$ *denotes the set of all runs of agent* Ag *that satisfy* Ψ*:*

$$\mathcal{R}_\Psi(Ag, Env) = \{r \mid r \in \mathcal{R}(Ag, Env) \text{ and } \Psi(r) = 1\}.$$

There are more possibilities to define the success of an agent in a task environment.

The pessimistic definition:

We say that an agent Ag succeeds in task environment (Env, Ψ) if $\mathcal{R}_\Psi(Ag, Env) = \mathcal{R}(Ag, Env)$.

Thus, the agent succeeds iff every possible run of the agent in the environment satisfies the predicate task specification.

Task environment

The optimistic definition:

We say that an agent Ag succeeds in task environment (Env, Ψ) if $\mathcal{R}_\Psi(Ag, Env) \neq \emptyset$.

Thus, the agent succeeds iff at least one run of the agent in the environment satisfies the predicate task specification.

The probabilistic definition:

The success of an agent Ag in task environment (Env, Ψ) is defined as the probability $P(\Psi | Ag, Env)$ that the predicate task specification Ψ is satisfied by the agent in the environment Env.

Remark 0.30

$P(\Psi | Ag, Env) = \sum_{r \in \mathcal{R}_\Psi(Ag, Env)} P(r | Ag, Env)$.

Achievement and maintenance tasks

- The notion of a predicate task specification may seem rather abstract.
- It is a generalization of certain very common forms of tasks.

Two most common types of tasks are achievement tasks and maintenance tasks:

- Achievement tasks are those of the form achieve state of affairs φ.
- Maintenance tasks are those of the form maintain state of affairs φ.

Achievement tasks

Definition 0.31

The task environment (Env, Ψ) specifies an achievement task if there exists some set of states $G \subseteq E$ such that for all $r \in \mathcal{R}(Ag, Env)$,

$\Psi(r) = 1$ iff there exists some state $e \in G$ such that e occurs in r.

We also say that (Env, Ψ) is an achievement task environment.

The elements of G are the goal states of the task.

Notation 0.32

We use (Env, G) to denote an achievement task environment with goal states G and environment Env.

An agent is successful if is guaranteed to bring about one of the goal states (we do not care which one — all are considered equally good).

Achievement tasks

A useful way to think about achievement tasks is as the agent playing a game against the environment:

- The environment and agent both begin in some state.
- The agent executes an action, and the environment responds with some state.
- The agent then executes another action, and so on.
- The agent wins if it can force the environment into one of the goal states.

Maintenance tasks

- Many other tasks can be classified as problems where the agent is required to avoid some state of affairs.
- We refer to such tasks as maintenance tasks.

Definition 0.33

The task environment (Env, Ψ) specifies a maintenance task if there exists some set of states $B \subseteq E$ such that for all $r \in \mathcal{R}(Ag, Env)$,

$$\Psi(r) = 1 \textit{ iff for any state } e \in B, \ e \textit{ does not occur in } r.$$

We also say that (Env, Ψ) is a maintenance task environment.

The elements of B are the bad states of the task.

Notation 0.34

We use (Env, B) to denote a maintenance task environment with bad states B and environment Env.

Maintenance tasks

It is again useful to think of maintenance tasks as games:

- The agent wins if it manages to avoid all the bad states.
- The environment, in the role of opponent, is attempting to force the agent into B.
- The agent is successful if it has a winning strategy for avoiding B.

Achievement and maintenance tasks

- More complex tasks might be specified by combinations of achievement and maintenance tasks.
- A simple combination:

  achieve any one of states G while avoiding all states B.

Logic-based agents

Logic-based agents

The logic-based approach is the classical approach to building agents.

Key ideas:

- give a symbolic representation of the environment - logical formulas.
- manipulate syntactically this representation - logical deduction or theorem proving.

Problems to be solved:

- Transduction problem: the problem of translating the real world into an accurate, adequate symbolic description of the world, in time for that description to be useful.
- Representation/reasoning problem: the problem of representing information symbolically, and getting agents to manipulate/reason with it, in time for the results to be useful.

Logic-based agents

The logic-based approach is the classical approach to building agents.

Key ideas:

- give a symbolic representation of the environment - logical formulas.
- manipulate syntactically this representation - logical deduction or theorem proving.

Problems to be solved:

- Transduction problem: the problem of translating the real world into an accurate, adequate symbolic description of the world, in time for that description to be useful.
- Representation/reasoning problem: the problem of representing information symbolically, and getting agents to manipulate/reason with it, in time for the results to be useful.

Logic-based agents

Deliberate agents are a simple model of logic-based agents.

- An internal state of such an agent is a database of formulas of first-order logic.
- The agent's database might contain formulas such as

  Open(valve1), *Temperature(reactor6, 32)*, *Pressure(tank6, 28)*.
- The database is the information that the agent has about its environment.
- An agent's database plays a somewhat analogous role to that of belief in humans.
- Some facts from the database could be wrong - agent's sensors may be faulty, its reasoning may be faulty, the information may be out of date.
- Thus, the fact that *Open(valve1)* is in the database does not mean that *valve1* is open; it could be closed.

Logic-based agents

We use the model of agents with state.
Let $\mathcal{L}$ be a first-order language and $Form_{\mathcal{L}}$ be the set of its formulas.

We assume that $\mathcal{L}$ contains:

- a unary relation symbol Do;
- a constant symbol c_α for every action $\alpha \in Ac$. For simplicity, we write α instead of c_α.

- A database is a set of formulas of $\mathcal{L}$.
- Let $\mathcal{D}$ be the set of all databases. Thus, $\mathcal{D} = 2^{Form_{\mathcal{L}}}$.
- We write $DB, DB_1, \ldots$ for members of $\mathcal{D}$.
- An internal state of the agent is a database. Thus, $I = \mathcal{D}$.

Logic-based agents

- We fix a set $\Sigma \subseteq Form_{\mathcal{L}}$ of formulas of $\mathcal{L}$, whose elements are called deduction formulas.
- We use the notation $DB \vdash_\Sigma \varphi$ for $DB \cup \Sigma \vdash \varphi$.
- The idea is that

  if $DB \vdash_\Sigma Do(\alpha)$, then α is the action to be performed by the agent.
- The agent's behaviour is determined by its deduction formulas (its program) and its current database.

An agent's action selection function

$$action : \mathcal{D} \to Ac$$

is defined in terms of its deduction formulas. The pseudo-code definition of this function is given in Figure 6.

Logic-based agents

```
1.   function action(DB : 𝒟) returns an action Ac
2.   begin
3.      for each α ∈ Ac do
4.         if DB ⊢_Σ Do(α) then
5.            return α
6.         end-if
7.      end-for
8.      for each α ∈ Ac do
9.         if DB ⊬_Σ ¬Do(α) then
10.           return α
11.        end-if
12.     end-for
13.     return null
14.  end function action
```

Figure 6: Agent selection as theorem proving.

Logic-based agents

- In lines 3-7, the agent takes each of its possible actions α in turn, and attempts to prove the formula $Do(\alpha)$ from its database DB (passed as a parameter to the function) using its set Σ of deduction formulas. If the agent succeeds in proving $Do(\alpha)$, then α is returned as the action to be performed.
- If the agent fails to prove $Do(\alpha)$, for all actions $\alpha \in Ac$, then it tries to find an action that is consistent with the deduction formulas and the database, that is not explicitly forbidden.
- In lines 8-12, the agent attempts to find an action $\alpha \in Ac$ such that $\neg Do(\alpha)$ cannot be derived from its database using its deduction formulas. If it can find such an action, then this is returned as the action to be performed.
- If, however, the agent fails to find an action that is at least consistent, then it returns the special action *null*, indicating that no action has been selected.

Logic-based agents

The perception function *see* remains unchanged:

$$see : E \to Per,$$

where *Per* is the set of percepts.

The *next* function has the form:

$$next : \mathcal{D} \times Per \to \mathcal{D}.$$

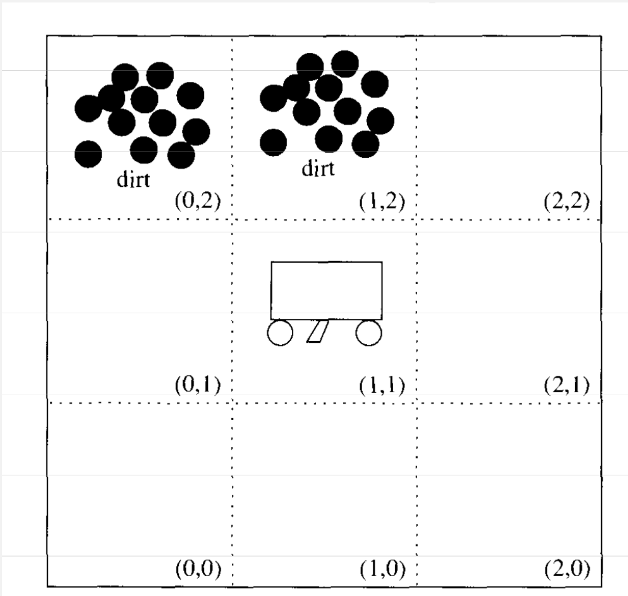
It maps a database and a percept to a new database.

Logic-based agents - example

We consider an example: vacuum cleaning world

- We have a small robotic agent that will clean up a house.
- The robot is equipped with a sensor that will tell it whether it is over any dirt, and a vacuum cleaner that can be used to suck up dirt.
- The robot always has a definite orientation (one of north, south, east, or west).
- In addition to being able to suck up dirt, the agent can move forward one 'step' or turn right 90 degrees.
- The agent moves around a room, which is divided grid-like into a number of equally sized squares.
- Our agent does nothing but clean — it never leaves the room.
- Assume, for simplicity, that the room is a 3×3 grid, and the agent always starts in grid square $(0, 0)$ facing north.

Logic-based agents - example



Logic-based agents - example

- The set *Per* of percepts is defined as
$$Per = \{dirt, nothing\},$$
where *dirt* signifies that there is dirt beneath it, and *nothing* indicates no special information.
- The set *Ac* of actions is defined as
$$Ac = \{forward, suck, turn\},$$
where *forward* means 'go forward', *suck* means 'suck up dirt', and *turn* means 'turn right 90 degrees'.
- The goal is to traverse the room continually searching for and removing dirt.

Logic-based agents - example

We use three simple domain predicates:

$In(i,j)$	agent is at (i,j),
$Dirt(i,j)$	there is dirt at (i,j),
$Facing(d)$	the agent is facing direction d,

where $i,j \in \{0,1,2\}$ and $d \in \{north, south, east, west\}$.

This means that the first-order language $\mathcal{L}$ contains:

- two binary relation symbols *In* and *Dirt*;
- a unary relation symbol *Facing*;
- constant symbols *north*, *south*, *east*, *west*;
- constant symbols (i,j) for every $i,j \in \{0,1,2\}$.

Logic-based agents - example

- The *next* function looks at the perceptual information obtained from the environment and at the actual database, and generates a new database which includes this information.
- It removes old or irrelevant information, and also, it tries to figure out the new location and orientation of the agent.
- We specify the *next* function in several parts.

Let old(DB) denote the set of 'old' information in a database, which we want the update function *next* to remove.

$$old(DB) = DB \cap \Delta,$$

where

$$\Delta = \{In(i,j) \mid i,j \in \{0,1,2\}\} \cup \{Dirt(i,j) \mid i,j \in \{0,1,2\}\} \cup \{Facing(d) \mid d \in \{north, south, east, west\}\}.$$

Logic-based agents - example

- We require a function new, which gives the set of new formulas to add to the database:

$$new : \mathcal{D} \times Per \to \mathcal{D}.$$

- It must generate formulas
  - $In(\ldots)$, describing the new position of the agent;
  - $Facing(\ldots)$ describing the orientation of the agent;
  - $Dirt(\ldots)$ if dirt has been detected at the new position.

The next function is defined as follows:

$$next : \mathcal{D} \times Per \to \mathcal{D}, \quad next(DB, p) = (DB - old(DB)) \cup new(DB, p)$$

Logic-based agents - example

The deduction formulas have the general form

$$\varphi \to \psi, \qquad \text{where } \varphi, \psi \text{ are formulas of } \mathcal{L}$$

Cleaning

$$In(x, y) \wedge Dirt(x, y) \to Do(suck) \qquad x, y \text{ are variables}$$

- If the agent is at location (x, y) and it perceives dirt, then the prescribed action will be to suck up dirt.
- It takes priority over all other possible behaviours of the agent (such as navigation).

Logic-based agents - example

Traversal

- The basic action of the agent is to traverse the world.
- For simplicity, we assume that the robot will always move from $(0, 0)$ to $(0, 1)$ to $(0, 2)$ and then to $(1, 2)$, $(1, 1)$ and so on. Once the agent reaches $(2, 2)$, it must head back to $(0, 0)$.
- The deduction formulas dealing with the traversal up to $(0, 2)$:

$$\begin{aligned}
In(0,0) \wedge Facing(north) \wedge \neg Dirt(0,0) &\to Do(forward) \\
In(0,1) \wedge Facing(north) \wedge \neg Dirt(0,1) &\to Do(forward) \\
In(0,2) \wedge Facing(north) \wedge \neg Dirt(0,2) &\to Do(turn) \\
In(0,2) \wedge Facing(east) &\to Do(forward)
\end{aligned}$$

- Similar formulas can be easily generated that will get the agent to $(2, 2)$ and back to $(0, 0)$.

Logic-based agents

Decision making is viewed as deduction, an agent's program is encoded as a logical theory, and actions selection reduces to a problem of proof.

Advantages:

- elegance and a clean (logical) semantics.

Disadvantages:

- inherent computational complexity of theorem proving;
- based on the assumption of calculative rationality:
  - world will not change in any significant way while the agent is deciding what to do;
  - an action which is rational when decision-making begins will be rational when it concludes.
- transduction and representation/reasoning problems essentially unsolved.