Notations for Agents

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

$E = \{e_1, e_2, e_3, \ldots \}$

Agents have a repertoire of possible actions (via effectors), which transform the state of the environment

$Ac = \{ \alpha_1, \alpha_2, \alpha_3, \ldots \}$

A run, $r$, of an agent in an environment is a sequence of interleaved environment states and actions:

$r : e_0 \rightarrow e_1 \rightarrow e_2 \rightarrow \cdots \rightarrow e_n$

$a = \text{action}, e = \text{environmental state}$

Environments

Formally, an environment, $Env$, is a triple where:

- $E$ is a set of environment states,
- $e_0$ is the initial state; and
- $\tau$ is a state transformer function.

$Env = < E, e_0, \tau >$

Let

- $\mathcal{R}$ be the set of all possible finite sequences
- $\mathcal{R}^{Ac}$ be the subset of these that end in actions
- $\mathcal{R}^E$ be the subset of all these that end in an environment state

Agents

- Agent is a function which maps runs to actions:

$Ag : \mathcal{R}^E \rightarrow Ac$

An agent makes a decision about what action to perform based on the history of the system that it has witnessed to date. Let $AG$ be the set of all agents

State Transformer Functions

- $\tau$ represents the behavior of the environment
- $\tau$ maps a run ending in an action to a set of environmental states

$\tau : \mathcal{R}^{Ac} \rightarrow \wp(E)$

Note that environments are . . .

- history dependent
- non-deterministic

- If $\tau(r) = \emptyset$ then there are no possible successor states to $r$ - the system has ended its run
Systems

- A **system** is a pair containing an agent and an environment.
- Any system will have associated with it a set of possible histories.
- We denote the set of histories of agent $Ag$ in environment $Env$ by $H(Ag, Env)$.
- We assume the system contains only terminated runs (no more actions are possible).

Histories

Formally, a sequence $(e_0, \alpha_0, e_1, \alpha_1, e_2, \alpha_2, \ldots)$ represents a history of an agent $Ag$ in an environment, $Env = < E, e_0, \tau >$ if:

1. $e_0$ is the initial state of $Env$.
2. $\alpha_0 = Ag(e_0)$; and
3. for all $u > 0$, $e_u \in \tau((e_0, \alpha_0, \ldots, e_u))$ where $\alpha_u = Ag((e_0, \alpha_0, \ldots, e_u))$.

Let:

- $R$ be the set of all such possible finite sequences (over $E$ and $Ac$).
- $R^{Ac}$ be the subset of these that end with an action.
- $R^E$ be the subset of these that end with an environment state.

Behaviorally Equivalent Agents

- Two agents $Ag_1$ and $Ag_2$ are *behaviorally equivalent* with respect to an environment $Env$ iff all runs produced by the agents in the environment are identical.

Perception and Action

- **see** function - an agent’s ability to observe its environment.
- **action** function - an agent’s decision-making process.
- Output of the **see** function is a percept: $see : E \rightarrow Per$ which maps environment states to percepts.
- **Action** is also a function which maps sequences of percepts to actions $action : Per^* \rightarrow A$.

The Perception System

![Perception System Diagram]
**See and Act**

```
environment
```

**Agent Control Loop**

1. Agent starts in some initial internal state $i_0$
2. Observes its environment state $e$, and generates a percept $see(e)$
3. Internal state of the agent is then updated via $next$ function, becoming $next(i_0, see(e))$
4. The action selected by the agent is $action(next(i_0, see(e)))$
5. Repeat from 2.

---

**Agents with State**

- Internal data structures save information about the environment state and history
- Let $I$ be the set of all internal states, $i_x$
- The $see$ function is the same, returns percept $see : E \rightarrow Per$
- The $action$ function now uses the internal state $action : I \rightarrow Ac$
- The internal state is updated by function $next$ $next : I \times Per \rightarrow I$

---

**Generic Agent with State Control Loop**

1. Start in some initial internal state $i_0$
2. Observes its environment state $e$, and receives a percept, $see (e)$
3. Internal state is updated via the $next$ function, $i_{k+1} = next(i_k, see(e))$
4. The $action$ selected by the agent is taken $action (next(i_k, see(e)))$
5. Repeat from 2

---

**Utility Functions over States**

- One possibility: associate utilities with individual states — the task of the agent is then to bring about states that maximize utility
- A task specification is a function $u : E \rightarrow \mathbb{R}$ which associates a real number with every environment state
Goal-Based Agent
- The agent tries to reach a desirable state
  - goal may be provided from the outside (user, designer), or inherent to the agent itself
- Results of possible actions are considered with respect to the goal
  - may require search or planning
- Very flexible action sequences produced
- Can be time-consuming (computationally)

Utility Functions over States
- But what is the value of a run...
  - minimum utility of state on run?
  - maximum utility of state on run?
  - sum of utilities of states on run?
  - average?
- Disadvantage: difficult to specify a long term view when assigning utilities to individual states
  (One possibility: a discount for states later on.)

Utilities over Runs
- Another possibility: assigns a utility not to individual states, but to runs themselves:
  \[ u : R \rightarrow \mathbb{R} \]
- Such an approach takes an inherently long term view
- Other variations: incorporate probabilities of different states emerging
- Difficulties with utility-based approaches:
  - where do the numbers come from?
  - we don’t think in terms of utilities!
  - hard to formulate tasks in these terms

How to Measure Utility of a Run?
- What is the utility value of a history?
  - Minimum utility of any state in the history?
  - Maximum utility of any state in the history?
  - Utility of final state?
  - Sum ...... ?
  - Average ......?
- Disadvantage: difficult to specify a long term view when assigning utilities to individual states.
- One possible solution: “discount” states later on.

Utility-Based Agent Program
- function Utility-Agent (percept)
  current world state
- static: rules, a set of condition-action rules
- state <- UPDATE-STATE (state, percept)
- do {
    rule <- RULE-MATCH (state, rules)
    action <- RULE-ACTION [rule]
    state <- UPDATE-STATE (state, action)
    } while ((not (goals? <- state)) || (not (good quality?)))
- return action

Utility in the Tileworld
- 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
- Holes have to be filled up with tiles by the agent. An agent scores points by filling holes with tiles, with the aim being to fill as many holes as possible
- TILEWORLD changes with the random appearance and disappearance of holes
- Utility function defined as follows:
  \[ U(r) = \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r} \]
**Expected Utility & Optimal Agents**

- Write $P(r \mid Ag, Env)$ to denote probability that run $r$ occurs when agent $Ag$ is placed in environment $Env$.

Note: $\sum_{r \in \mathcal{R}(Ag, Env)} P(r \mid Ag, Env) = 1$.

- Then optimal agent $Ag_{opt}$ in an environment $Env$ is the one that

$$Ag_{opt} = \arg \max_{Ag \in Ags} \sum_{r \in \mathcal{R}(Ag, Env)} u(r) P(r \mid Ag, Env).$$

*(1)*

**Summary**

- Notations for agents
- Abstract architectures
- Agent decision making
- Example Concrete Architectures
- Reference –
  - Wooldridge MAS, Ch. 2
  - Weiss, Ch. 1.3, 1.4, 1.5
  - Russell and Norvig AIMA, Ch. 2