Intelligent Agent

Agent Architecture
Agent Architectures

• An *agent* is a computer system capable of *flexible autonomous action*…

• Three types of agent *architecture*:
  – symbolic/logical
  – reactive
  – hybrid
Agent Architectures

• We want to build agents, that enjoy the properties of autonomy, reactivity, pro-activity, and social ability that we talked about earlier.
• This is the area of agent architectures.
• Maes defines an agent architecture as:
  ‘[A] particular methodology for building [agents]. It specifies how... the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. The total set of modules and their interactions has to provide an answer to the question of how the sensor data and the current internal state of the agent determine the actions... and future internal state of the agent. An architecture encompasses techniques and algorithms that support this methodology.’
Agent Architectures

- Originally (1956-1985), pretty much all agents designed within AI were *symbolic reasoning* agents.
- Its purest expression proposes that agents use *explicit logical reasoning* in order to decide what to do.
- Problems with symbolic reasoning led to a reaction against this — the so-called *reactive agents* movement, 1985–present.
- From 1990-present, a number of alternatives proposed: *hybrid* architectures, which attempt to combine the best of reasoning and reactive architectures.
Abstract Architecture for Agents

- Assume the environment may be in any of a finite set $E$ of discrete, instantaneous states:
  
  $$ E = \{e, e', \ldots\}. $$

- Agents are assumed to have a repertoire of possible actions available to them, which transform the state of the environment:
  
  $$ Ac = \{\alpha, \alpha', \ldots\} $$

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

  $$ r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} e_3 \xrightarrow{\alpha_3} \cdots \xrightarrow{\alpha_{u-1}} e_u $$
Abstract Architecture for Agents

• 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
State Transformer Functions

• A *state transformer* function represents behavior of the environment:

\[
\tau : \mathcal{R}^A_c \rightarrow \wp(E)
\]

• Note that environments are…
  – *history dependent*
  – *non-deterministic*

• If \(\tau(r) = \emptyset\), then there are no possible successor states to \(r\). In this case, we say that the system has *ended* its run.

• Formally, we say an environment \(Env\) is a triple \(Env = \langle E, e_0, \tau \rangle\) where: \(E\) is a set of environment states, \(e_0 \in E\) is the initial state, and \(\tau\) is a state transformer function.
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 \( \text{AG} \) be the set of all agents.
Systems

- A *system* is a pair containing an agent and an environment.
- Any system will have associated with it a set of possible runs; we denote the set of runs of agent $Ag$ in environment $Env$ by $R(Ag, Env)$.
- (We assume $R(Ag, Env)$ contains only *terminated* runs.)
Systems

- Formally, a sequence

\[(e_0, \alpha_0, e_1, \alpha_1, e_2, \ldots)\]

represents a run of an agent \(Ag\) in environment \(Env = \langle E, e_0, \tau \rangle\) if:

1. \(e_0\) is the initial state of \(Env\)
2. \(\alpha_0 = Ag(e_0)\); and
3. For \(u > 0\),

\[e_u \in \tau((e_0, \alpha_0, \ldots, \alpha_{u-1}))\]

where

\[\alpha_u = Ag((e_0, \alpha_0, \ldots, e_u))\]
Purely Reactive Agents

- Some agents decide what to do without reference to their history — they base their decision making entirely on the present, with no reference at all to the past
- We call such agents purely reactive:
  \[ \text{action} : E \rightarrow Ac \]
- A thermostat is a purely reactive agent
  \[
  \text{action}(e) = \begin{cases} 
  \text{off} & \text{if } e = \text{temperature OK} \\
  \text{on} & \text{otherwise.}
  \end{cases}
  \]
• Now introduce *perception* system:

![Diagram]

- **Environment**
- **Agent**
  - See
  - Action
Perception

• The see function is the agent’s ability to observe its environment, whereas the action function represents the agent’s decision making process.

• Output of the see function is a percept:

\[ \text{see} : E \rightarrow \text{Per} \]

which maps environment states to percepts, and action is now a function

\[ \text{action} : \text{Per}^* \rightarrow A \]

which maps sequences of percepts to actions.

• Example of indistinguishable environments states via see
Agents with State

• We now consider agents that maintain state:
Agents with State

- These agents have some internal data structure, which is typically used to record information about the environment state and history. Let $I$ be the set of all internal states of the agent.
- The perception function $\text{see}$ for a state-based agent is unchanged:
  \[ \text{see} : E \rightarrow \text{Per} \]
  The action-selection function $\text{action}$ is now defined as a mapping
  \[ \text{action} : I \rightarrow \text{Ac} \]
  from internal states to actions. An additional function $\text{next}$ is introduced, which maps an internal state and percept to an internal state:
  \[ \text{next} : I \times \text{Per} \rightarrow I \]
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. Goto 2
Tasks for Agents

- We build agents in order to carry out *tasks* for us
- The task must be *specified* by us…
- But we want to tell agents what to do *without* telling them how to do it
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
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:

\[ u : R \to IR \]

• 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
The Tileworld, Some Examples

- From Goldman and Rosenschein, AAAI-94:

![Diagram of the Tileworld]

**Figure 1: Strongly-Coupled Interactions**
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. This requires:
  - React to changes in the environment
  - Exploit opportunities when they arise
- 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 $Env$.

  \[ \sum_{r \in \mathcal{R}(Ag, Env)} P(r \mid Ag, Env) = 1. \]

  Note:

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

  \[ Ag_{opt} = \arg \max_{Ag \in \mathcal{A}} \sum_{r \in \mathcal{R}(Ag, Env)} u(r)P(r \mid Ag, Env). \] (1)
Bounded Optimal Agents

- Some agents cannot be implemented on some computers (A function $A_g : \mathbb{R}^E \rightarrow A_c$ may need more than available memory to implement).
- Write $AG_m$ to denote the agents that can be implemented on machine (computer) $m$:
  $$AG_m = \{A_g \mid A_g \in AG \text{ and } A_g \text{ can be implemented on } m\}.$$
- We can replace equation (1) with the following, which defines the bounded optimal agent $A_{g_{opt}}$:
  $$A_{g_{opt}} = \arg \max_{A_g \in AG_m} \sum_{r \in R(A_g, \text{Env})} u(r)P(r \mid A_g, \text{Env}).$$ (2)
Predicate Task Specifications

• A special case of assigning utilities to histories is to assign 0 (false) or 1 (true) to a run.
• If a run is assigned 1, then the agent succeeds on that run, otherwise it fails.
• Call these *predicate task specifications*.
• Denote predicate task specification by $\Psi$. Thus $\Psi : R \rightarrow \{0, 1\}$. 
Task Environments

- A *task environment* is a pair $\langle Env, \Psi \rangle$ where $Env$ is an environment,

  $$\Psi : \mathbb{R} \rightarrow \{0, 1\}$$

  is a predicate over runs.

  Let $TE$ be the set of all task environments.

- A task environment specifies:
  - the properties of the system the agent will inhabit
  - the criteria by which an agent will be judged to have either failed or succeeded
Task Environments

• Write $\mathcal{R}_\Psi(Ag, Env)$ to denote set of all runs of the agent $Ag$ in environment $Env$ that satisfy $\Psi$:

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

• We then say that an agent $Ag$ succeeds in task environment $\langle Env, \Psi \rangle$ if

$$\mathcal{R}_\Psi(Ag, Env) = \mathcal{R}(Ag, Env).$$
The Probability of Success

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

- Then the probability $P(\Psi \mid Ag, Env)$ that $\Psi$ is satisfied by $Ag$ in $Env$ would then simply be:

$$P(\Psi \mid Ag, Env) = \sum_{r \in \mathcal{R}_{\Psi}(Ag, Env)} P(r \mid Ag, Env)$$
Achievement & Maintenance Tasks

- Two most common types of tasks are *achievement tasks* and *maintenance tasks*:
  1. *Achievement tasks* are those of the form “achieve state of affairs $\phi$”
  2. *Maintenance tasks* are those of the form “maintain state of affairs $\psi$”
 Achievement & Maintenance Tasks

• An achievement task is specified by a set $G$ of “good” or “goal” states: $G \subseteq E$
  The agent succeeds if it is guaranteed to bring about at least one of these states (we do not care which one — they are all considered equally good).

• A maintenance goal is specified by a set $B$ of “bad” states: $B \subseteq E$
  The agent succeeds in a particular environment if it manages to avoid all states in $B$ — if it never performs actions which result in any state in $B$ occurring.
Agent Synthesis

• *Agent synthesis* is automatic programming: goal is to have a program that will take a task environment, and from this task environment automatically generate an agent that succeeds in this environment:

\[
\text{syn} : \mathcal{T}E \rightarrow (\mathcal{A}G \cup \{\perp\}).
\]

(Think of $\perp$ as being like *null* in Java.)

• Synthesis algorithm is:
  
  – *Sound* (đúng đắn) if, whenever it returns an agent, then this agent succeeds in the task environment that is passed as input
  
  – *Complete* (đầy đủ) if it is guaranteed to return an agent whenever there exists an agent that will succeed in the task environment given as input
Agent Synthesis

• Synthesis algorithm $syn$ is sound if it satisfies the following condition:

$$syn(\langle Env, \Psi \rangle) = Ag \implies \mathcal{R}(Ag, Env) = \mathcal{R}_\Phi(Ag, Env).$$

and complete if:

$$\exists Ag \in \mathcal{AG} \text{ s.t. } \mathcal{R}(Ag, Env) = \mathcal{R}_\Phi(Ag, Env) \implies syn(\langle Env, \Psi \rangle) \neq \bot.$$