Agent-Based Systems

Michael Rovatsos
mrovatso@inf.ed.ac.uk

Lecture 2 – Abstract Agent Architectures
Where are we?

Last time . . .
  - Introduced basic and advanced aspects of agency
  - Situatedness, autonomy and environments
  - Reactivity, proactiveness and social ability
  - Compared agents to other types of systems

Today . . .
  - Abstract Agent Architectures
Purpose of this lecture: formalise what we have discussed so far
Will result in an abstract specification of agents
Not about concrete agent architectures which we can actually implement (but see later)
Assume a discrete, finite set of environment states \( E = \{ e, e', \ldots \} \) (or approximation of continuous state space)
Assume action repertoire of agents is defined by \( Ac = \{ \alpha, \alpha', \ldots \} \)
Idea: environment starts at some state and agent chooses action in each state which leads to new (set of) state(s)
Abstract agent architectures

- **Run** = sequence of interleaved environment states and actions

\[ r : e_0 \xrightarrow{\alpha_0} e_1 \xrightarrow{\alpha_1} e_2 \xrightarrow{\alpha_2} \cdots e_{u-1} \xrightarrow{\alpha_{u-1}} e_u \]

- Define \( \mathcal{R} = \{r, r', \ldots\} \) the set of all such possible finite sequences

- \( \mathcal{R}^{Ac}/\mathcal{R}^E \) subsets of \( \mathcal{R} \) that end with an action/environment state

- **State transformer function** is a function \( \tau : \mathcal{R}^{Ac} \rightarrow \wp(E) \)

- \( \tau \) maps each run ending with an agent action to the set of possible resulting states
  - Depends on history of previous states
  - Uncertainty/non-determinism modelled by allowing for multiple successor states

- If \( \tau(r) = \emptyset \) system terminates (we assume it always will eventually)
Abstract agent architectures

- Next, we have to specify how agent functions
- Agents choose actions depending on states
- In contrast to environments, we assume them to be deterministic
- In the most general sense an agent is a function
  \[ Ag : \mathcal{R}^E \rightarrow Ac \]
- If set of all agents is \( \mathcal{AG} \), define system as pair of an agent \( Ag \) and an environment \( Env \)
- Denote runs of system by \( \mathcal{R}(Ag, Env) \) and assume they are all terminal (and thus finite)
Abstract agent architectures

- A sequence \((e_0, \alpha_0, e_1, \alpha_1, \ldots)\) represents a run of agent \(Ag\) in environment \(Env = \langle E, e_0, \tau \rangle\) if
  1. \(e_0\) is initial state of \(E\)
  2. \(\alpha_0 = Ag(e_0)\)
  3. For \(u > 0\)
     \[ e_u \in \tau((e_0, \alpha_0, e_1, \ldots, \alpha_{u-1})) \]
     and
     \[ \alpha_u = Ag((e_0, \alpha_0, e_1, \ldots, e_u)) \]

- Two agents \(Ag_1\) and \(Ag_2\) are called **behaviourally equivalent with respect to environment** \(Env\) iff
  \[ R(Ag_1, Env) = R(Ag_2, Env) \]

- If this is true for any environment \(Env\), they are simply called **behaviourally equivalent**
Purely reactive agents

- Pure reactivity means basing decisions only on present state
- History is not taken into account
- “Behaviourist” model of activity: actions are based on stimulus-response schemata
- Formally they are described by a function

\[ Ag : E \rightarrow Ac \]

- Every purely reactive agent can be mapped to an agent defined on runs (the reverse is usually not true)
- Example: thermostat with two environment states

\[ Ag(e) = \begin{cases} 
\text{heater off} & \text{if } e = \text{temperature OK} \\
\text{heater on} & \text{else}
\end{cases} \]
Agent-Based Systems

Perception and action

- Model so far is easy, but more design choices have to be made to turn it into more concrete **agent architectures**
- Agent architectures describe the internal structure of an agent (data structures, operations on them, control flow)
- First steps: define **perception** and **action** subsystems
- Define functions $\text{see} : E \rightarrow \text{Per}$ and $\text{action} : \text{Per}^* \rightarrow \text{Ac}$ where
  - $\text{Per}$ is a non-empty set of percepts that the agent can obtained through its sensors
  - $\text{see}$ describes this process of perception and $\text{action}$ defines decisions based on percept sequences
- Agent definition now becomes $\text{Ag} = \langle \text{see}, \text{action} \rangle$
Perception and action

- If $e_1 \neq e_2 \in E$ and $\text{see}(e_1) = \text{see}(e_2)$ we call $e_1$ and $e_2$ indistinguishable
- Let $x =$ “the room temperature is OK” and $y =$ “Tony Blair is Prime Minister” be the only two facts that describe environment
- Then we have $E = \{\{\neg x, \neg y\}, \{\neg x, y\}, \{x, \neg y\}, \{x, y\}\}$
- If percepts of thermostat are $p_1$ (too cold) and $p_2$ (OK), indistinguishable states occur (unless PM makes room chilly)

$$\text{see}(e) = \begin{cases} p_1 & \text{if } e = e_1 \lor e = e_2 \\ p_2 & \text{if } e = e_3 \lor e = e_4 \end{cases}$$

- We write $e \sim e'$ (equivalence relation over states)
- The coarser these equivalence classes, the less effective is perception (if $| \sim | = |E|$ agent is omniscient)
Agents with state

- Mapping from runs to actions somewhat counter-intuitive
- We should rather think of agents as having **internal states** to reflect the internal representation they have of themselves and their environment
- Assuming an agent has a set $I$ of internal states, we can define its abstract architecture as follows:

\[
\text{see} : E \rightarrow \text{Per} \\
\text{action} : I \rightarrow \text{Ac} \\
\text{next} : I \times \text{Per} \rightarrow I
\]

- Behaviour: If initial internal state is $i$,
  - Observe environment, obtain $\text{see}(e)$
  - Update internal state to be $i' \leftarrow \text{next}(i, \text{see}(e))$
  - Action selection given by $\text{action}(i')$
  - Enter next cycle with $i \leftarrow i'$
Fundamental aspect of autonomy:

We want to tell agent what to do, but not how to do it

After all, this is what we want to be different from systems not based on intelligent agents

Roughly speaking, we can specify

- task to perform
- (set of) goal state(s) to be reached
- to maximise some performance measure

We start with the latter, which is based on utilities associated with states
Utilities

- Utilities describe “quality” of a state through some numerical value
- Doesn’t specify how to reach preferred states
- **Utility functions**: \( u : E \rightarrow \mathbb{R} \)
  - Using this, we can define overall utility of an agent to be
    - Worst utility of visited states (pessimistic)
    - Best utility of visited states (optimistic)
    - Average utility of visited states
    - …
- Disadvantage: long-term view is difficult to take into account
- We can use runs instead: \( u : \mathcal{R} \rightarrow \mathbb{R} \)
Optimal agents

- Assuming the utility function $u$ is bounded (i.e. $\exists k \in \mathbb{R} \ \forall r \in \mathcal{R} \ . \ u(r) \leq k$) we can define what optimal agents are:

> An optimal agent is one that maximises expected utility (MEU principle)

- To define this, assume $P(r|Ag, Env)$ is the probability that run $r$ occurs when agent $Ag$ is operating in environment $Env$.

- For optimal agent, the following equation holds:

$$Ag_{opt} = \arg \max_{Ag \in \mathcal{A}G} \sum_{r \in \mathcal{R}(Ag, Env)} P(r|Ag, Env) u(r)$$

- Often notion of bounded optimal agent is more useful, since not any function $Ag : \mathcal{R}^E \rightarrow Ac$ can be implemented on any machine.

- Define $\mathcal{A}G_m = \{ Ag | Ag \in \mathcal{A}G \text{can be implemented on machine } m \}$ and restrict maximisation to $\mathcal{A}G_m$ above.
Predicate task specifications

- Often more natural to define a predicate over runs (idea of success and failure)
- Assume \( u \) ranges over \( \{0, 1\} \), run \( r \in \mathcal{R} \) satisfies a task specification if \( u(r) = 1 \) (fails, else)
- Define: \( \Psi(r) \) iff \( u(r) = 1 \) and a task environment \( \langle Env, \Psi \rangle \) with \( \mathcal{T} \mathcal{E} \) the set of all task environments
- Further, let \( \mathcal{R}_\Psi(Ag, Env) = \{ r | r \in \mathcal{R}(Ag, Env) \land \Psi(r) \} \) the set of runs of agent \( Ag \) that satisfy \( \Psi \)
  - \( Ag \) succeeds in task environment \( \langle Env, \Psi \rangle \) iff \( \mathcal{R}_\Psi(Ag, Env) = \mathcal{R}(Ag, Env) \)
  - Quite demanding (pessimistic), we may require instead that there exists such a run (\( \exists r \in \mathcal{R}(Ag, Env) \cdot \Psi(r) \))
- We can extend state transformer function \( \tau \) by probabilities and require that \( P(\Psi|Ag, Env) = \sum_{r \in \mathcal{R}_\Psi(Ag,Env)} P(r|Ag, Env) \)
Achievement and maintenance tasks

- Two very common types of tasks:
  - “achieve state of affairs $\varphi$”
  - “maintain state of affairs $\varphi$”

- Achievement tasks are defined by a set of goal states
- Formally: $\langle Env, \Psi \rangle$ is an achievement task iff
  \[
  \exists G \subseteq E \ \forall r \in \mathcal{R}(Ag, Env) . \Psi(r) \iff \exists e \in G . e \in r
  \]

- Maintenance tasks are about avoiding certain failure states
- Formally: $\langle Env, B \rangle$ is a maintenance task iff
  \[
  \exists B \subseteq E \ \forall r \in \mathcal{R}(Ag, Env) . \Psi(r) \iff \forall e \in B . e \notin r
  \]

- There also exist more complex combinations of these
Summary

- Discussed abstract agent architectures
- Environments, perception & action
- Purely reactive agents, agents with state
- Utility-based agents
- Task-based agents, achievement/maintenance tasks
- Next time: Deductive Reasoning Agents