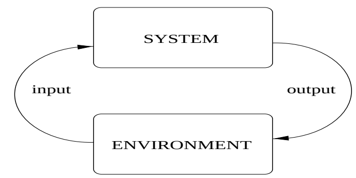
LECTURE 2: INTELLIGENT AGENTS

An Introduction to Multiagent Systems

http://www.csc.liv.ac.uk/~mjw/pubs/imas/

1 What is an Agent?

- The main point about agents is they are *autonomous*: capable of acting independently, exhibiting control over their internal state.
- Thus: *an agent is a computer system capable of* autonomous action *in some environment*.



- Trivial (non-interesting) agents:
  - thermostat;
  - UNIX daemon (e.g., biff).
- *An intelligent agent is a computer system capable of* flexible *autonomous action in some environment*.

  By *flexible*, we mean:
  - *reactive*;
  - *pro-active*;
  - *social*.

1.1 Reactivity

- If a program's environment is guaranteed to be fixed, the program need never worry about its own success or failure — program just executes blindly.

  Example of fixed environment: compiler.
- The real world is not like that: things change, information is incomplete. Many (most?) interesting environments are *dynamic*.
- Software is hard to build for dynamic domains: program must take into account possibility of failure — ask itself whether it is worth executing!
- A *reactive* system is one that maintains an ongoing interaction with its environment, and responds to changes that occur in it (in time for the response to be useful).

1.2 Proactiveness

- Reacting to an environment is easy (e.g., stimulus → response rules).
- But we generally want agents to *do things for us*.
- Hence *goal directed behaviour*.
- Pro-activeness = generating and attempting to achieve goals; not driven solely by events; taking the initiative.
- Recognising opportunities.

1.3 Social Ability

- The real world is a *multi*-agent environment: we cannot go around attempting to achieve goals without taking others into account.
- Some goals can only be achieved with the cooperation of others.
- Similarly for many computer environments: witness the INTERNET.
- *Social ability* in agents is the ability to interact with other agents (and possibly humans) via some kind of *agent-communication language*, and perhaps cooperate with others.

2 Other Properties

- Other properties, sometimes discussed in the context of agency:
  - *mobility*: the ability of an agent to move around an electronic network;
  - *veracity*: an agent will not knowingly communicate false information;
  - *benevolence*: agents do not have conflicting goals, and that every agent will therefore always try to do what is asked of it;
  - *rationality*: agent will act in order to achieve its goals, and will not act in such a way as to prevent its goals being achieved — at least insofar as its beliefs permit;
  - *learning/adaption*: agents improve performance over time.

2.1 Agents and Objects

- Are agents just objects by another name?
- Object:
  - encapsulates some state;
  - communicates via message passing;
  - has methods, corresponding to operations that may be performed on this state.

- Main differences:
  - *agents are autonomous*:
    agents embody stronger notion of autonomy than objects, and in particular, they decide for themselves whether or not to perform an action on request from another agent;
  - *agents are smart*:
    capable of flexible (reactive, pro-active, social) behavior, and the standard object model has nothing to say about such types of behavior;
  - *agents are active*:
    a multi-agent system is inherently multi-threaded, in that each agent is assumed to have at least one thread of active control.

Objects do it for free...

- *agents do it because they want to;*
- *agents do it for money*.

2.2 Agents and Expert Systems

- Aren't agents just expert systems by another name?
- Expert systems typically disembodied 'expertise' about some (abstract) domain of discourse (e.g., blood diseases).
- Example: MYCIN knows about blood diseases in humans.

  It has a wealth of knowledge about blood diseases, in the form of rules.

  A doctor can obtain expert advice about blood diseases by giving MYCIN facts, answering questions, and posing queries.

- Main differences:
  - agents *situated in an environment*:
    MYCIN is not aware of the world — only information obtained is by asking the user questions.
  - agents *act*:
    MYCIN does not operate on patients.
- Some *real-time* (typically process control) expert systems *are* agents.

2.3 Intelligent Agents and AI

- Aren't agents just the AI project?
  Isn't building an agent what AI is all about?
- AI aims to build systems that can (ultimately) understand natural language, recognise and understand scenes, use common sense, think creatively, etc — all of which are very hard.
- So, don't we need to solve all of AI to build an agent. . . ?

- When building an agent, we simply want a system that can choose the right action to perform, typically in a limited domain.
- We *do not* have to solve *all* the problems of AI to build a useful agent:

  a little intelligence goes a long way!

- Oren Etzioni, speaking about the commercial experience of NETBOT, Inc:

  We made our agents dumber and dumber and dumber
  . . . until finally they made money.

3 Environments

- *Accessible* vs *inaccessible*.
  An accessible environment is one in which the agent can obtain complete, accurate, up-to-date information about the environment's state.
  Most moderately complex environments (including, for example, the everyday physical world and the Internet) are inaccessible.
  The more accessible an environment is, the simpler it is to build agents to operate in it.

- *Deterministic* vs *non-deterministic*.
  As we have already mentioned, a deterministic environment is one in which any action has a single guaranteed effect — there is no uncertainty about the state that will result from performing an action.
  The physical world can to all intents and purposes be regarded as non-deterministic.
  Non-deterministic environments present greater problems for the agent designer.

- *Episodic* vs *non-episodic*.

  In an episodic environment, the performance of an agent is dependent on a number of discrete episodes, with no link between the performance of an agent in different scenarios.

  Episodic environments are simpler from the agent developer's perspective because the agent can decide what action to perform based only on the current episode — it need not reason about the interactions between this and future episodes.

- *Static* vs *dynamic*.

  A static environment is one that can be assumed to remain unchanged except by the performance of actions by the agent.

  A dynamic environment is one that has other processes operating on it, and which hence changes in ways beyond the agent's control.

  The physical world is a highly dynamic environment.

- *Discrete* vs *continuous*.

  An environment is discrete if there are a fixed, finite number of actions and percepts in it. Russell and Norvig give a chess game as an example of a discrete environment, and taxi driving as an example of a continuous one.

4 Agents as Intentional Systems

- When explaining human activity, it is often useful to make statements such as the following:

  > Janine took her umbrella because she *believed* it was going to rain.
  > Michael worked hard because he *wanted* to possess a PhD.

- These statements make use of a *folk psychology*, by which human behaviour is predicted and explained through the attribution of *attitudes*, such as believing and wanting (as in the above examples), hoping, fearing, and so on.
- The attitudes employed in such folk psychological descriptions are called the *intentional* notions.

- The philosopher Daniel Dennett coined the term *intentional system* to describe entities 'whose behaviour can be predicted by the method of attributing belief, desires and rational acumen'.
- Dennett identifies different 'grades' of intentional system:

  'A *first-order* intentional system has beliefs and desires (etc.) but no beliefs and desires *about* beliefs and desires. . . . A *second-order* intentional system is more sophisticated; it has beliefs and desires (and no doubt other intentional states) about beliefs and desires (and other intentional states) — both those of others and its own'.
- Is it legitimate or useful to attribute beliefs, desires, and so on, to computer systems?

- McCarthy argued that there are occasions when the *intentional stance* is appropriate:

'To ascribe *beliefs*, *free will*, *intentions*, *consciousness*, *abilities*, or *wants* to a machine is *legitimate* when such an ascription expresses the same information about the machine that it expresses about a person. It is *useful* when the ascription helps us understand the structure of the machine, its past or future behaviour, or how to repair or improve it. It is perhaps never *logically required* even for humans, but expressing reasonably briefly what is actually known about the state of the machine in a particular situation may require mental qualities or qualities isomorphic to them. Theories of belief, knowledge and wanting can be constructed for machines in a simpler setting than for humans, and later applied to humans. Ascription of mental qualities is *most straightforward* for machines of known structure such as thermostats and computer operating systems, but is *most useful* when applied to entities whose structure is incompletely known'.

- What objects can be described by the intentional stance?
- As it turns out, more or less anything can. . . consider a light switch:

  'It is perfectly coherent to treat a light switch as a (very cooperative) agent with the capability of transmitting current at will, who invariably transmits current when it believes that we want it transmitted and not otherwise; flicking the switch is simply our way of communicating our desires'. (Yoav Shoham)
- But most adults would find such a description absurd!
  Why is this?

- The answer seems to be that while the intentional stance description is consistent,

  . . . it does not *buy us anything*, since we essentially understand the mechanism sufficiently to have a simpler, mechanistic description of its behaviour. (Yoav Shoham)
- Put crudely, the more we know about a system, the less we need to rely on animistic, intentional explanations of its behaviour.
- But with very complex systems, a mechanistic, explanation of its behaviour may not be practicable.
- *As computer systems become ever more complex, we need more powerful abstractions and metaphors to explain their operation — low level explanations become impractical.*
  The intentional stance is such an abstraction.

- The intentional notions are thus *abstraction tools*, which provide us with a convenient and familiar way of describing, explaining, and predicting the behaviour of complex systems.
- Remember: most important developments in computing are based on new *abstractions*:
  - procedural abstraction;
  - abstract data types;
  - objects.

  Agents, and agents as intentional systems, represent a further, and increasingly powerful abstraction.
- So agent theorists start from the (strong) view of agents as intentional systems: one whose simplest consistent description requires the intentional stance.

- This *intentional stance* is an *abstraction tool* — a convenient way of talking about complex systems, which allows us to predict and explain their behaviour without having to understand how the mechanism actually works.
- Now, much of computer science is concerned with looking for abstraction mechanisms (witness procedural abstraction, ADTs, objects, . . .)

  So why not use the intentional stance as an abstraction tool in computing — to explain, understand, and, crucially, program computer systems?
- This is an important argument in favour of agents.

- Other 3 points in favour of this idea:

Characterising Agents

- It provides us with a familiar, non-technical way of *understanding & explaing* agents.

Nested Representations

- It gives us the potential to specify systems that *include representations of other systems*.

  It is widely accepted that such nested representations are essential for agents that must cooperate with other agents.

Post-Declarative Systems

- This view of agents leads to a kind of post-declarative programming:
  - in procedural programming, we say exactly *what* a system should do;
  - in declarative programming, we state something that we want to achieve, give the system general info about the relationships between objects, and let a built-in control mechanism (e.g., goal-directed theorem proving) figure out what to do;
  - with agents, we give a very abstract specification of the system, and let the control mechanism figure out what to do, knowing that it will act in accordance with some built-in theory of agency (e.g., the well-known Cohen-Levesque model of intention).

An aside...

- We find that researchers from a more mainstream computing discipline have adopted a similar set of ideas...
- In distributed systems theory, *logics of knowledge* are used in the development of *knowledge based protocols*.
- The rationale is that when constructing protocols, one often encounters reasoning such as the following:

  IF process i knows process j has received message m_1
  THEN process i should send process j the message m_2.

- In DS theory, knowledge is *grounded* — given a precise interpretation in terms of the states of a process; return to this later... we'll examine this point in detail later.

5 Abstract Architectures 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:

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

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

State Transformer Functions

- A *state transformer* function represents behaviour of the environment:

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

- 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 τ is a state transformer function.

Agents

- Agent is a function which maps runs to actions:

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

An agent makes a decision about what action to perform based on the history of the system that it has witnessed to date. Let $\mathcal{AG}$ 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 $\mathcal{R}(Ag, Env)$.
- (We assume $\mathcal{R}(Ag, Env)$ contains only *terminated* runs.)

- 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 $u > 0$,

$$\begin{aligned} e_u &\in \tau((e_0, \alpha_0, \ldots, \alpha_{u-1})) \quad \text{where} \\ \alpha_u &= Ag((e_0, \alpha_0, \ldots, e_u)) \end{aligned}$$

6 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*:

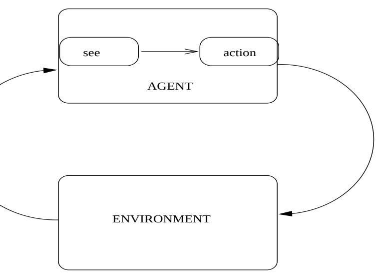
$$action : E \to Ac$$

- A thermostat is a purely reactive agent.

$$action(e) = \begin{cases} \text{off if } e = \text{temperature OK} \\ \text{on otherwise.} \end{cases}$$

7 Perception

- Now introduce *perception* system:



- 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*:

$$see : E \to Per$$

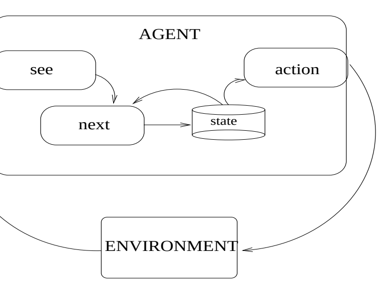
which maps environment states to percepts, and *action* is now a function

$$action : Per^* \to A$$

which maps sequences of percepts to actions.

8 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.

  Let I be the set of all internal states of the agent.
- The perception function *see* for a state-based agent is unchanged:

$$see : E \to Per$$

The action-selection function *action* is now defined as a mapping

$$action : I \to Ac$$

from internal states to actions. An additional function *next* is introduced, which maps an internal state and percept to an internal state:

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

8.1 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 $action(next(i_0, see(e)))$. This action is then performed.
5. Goto (2).

9 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.

9.1 Utilities Functions over States

- One possibility: associate *utilities* with individual states — the task of the agent is then to bring about states that maximise utility.
- A task specification is a function

$$u : E \to I\!R$$

which associated a real number with every environment state.

- 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.)

9.2 Utilities over Runs

- Another possibility: assigns a utility not to individual states, but to runs themselves:

$$u : \mathcal{R} \rightarrow I\!R$$

- 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.

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:

$$u(r) \hat{=} \frac{\text{number of holes filled in } r}{\text{number of holes that appeared in } r}$$

9.3 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 *maximizes expected utility*:

$$Ag_{opt} = \arg \max_{Ag \in \mathcal{AG}} \sum_{r \in \mathcal{R}(Ag, Env)} u(r) P(r \mid Ag, Env). \quad (1)$$

9.4 Bounded Optimal Agents

- Some agents cannot be implemented on some computers

  (A function $Ag : \mathcal{R}^E \rightarrow Ac$ may need more than available memory to implement.)
- Write $\mathcal{AG}_m$ to denote the agents that can be implemented on machine (computer) m:

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

- We can replace equation (1) with the following, which defines the *bounded optimal* agent Ag_{opt}:

$$Ag_{opt} = \arg \max_{Ag \in \mathcal{AG}_m} \sum_{r \in \mathcal{R}(Ag, Env)} u(r) P(r \mid Ag, Env). \quad (2)$$

9.5 Predicate Task Specifi cations

- 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 Ψ.
  Thus $\Psi : \mathcal{R} \to \{0, 1\}$.

9.6 Task Environments

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

$$\Psi : \mathcal{R} \to \{0, 1\}$$

  is a predicate over runs.
  Let $\mathcal{TE}$ 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.

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

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

- 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 Ψ 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 ϕ".
  2. *Maintenance tasks* Are those of the form "maintain state of affairs ψ".

- 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.

10 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:

$$syn : \mathcal{TE} \to (\mathcal{AG} \cup \{\bot\}).$$

(Think of $\bot$ as being like `null` in JAVA.

- Synthesis algorithm is:
  – *sound* if, whenever it returns an agent, then this agent succeeds in the task environment that is passed as input; and
  – *complete* if it is guaranteed to return an agent whenever there exists an agent that will succeed in the task environment given as input.

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

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

and complete if:

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