Utility functions and Concrete architectures: deductive agents

CS7032: AI for IET

Lecturer: Liliana Mamani Sanchez
mamanisl@cs.tcd.ie
October 12, 2015

Abstract Architectures (ctd.)

• An agent’s behaviour is encoded as its history:
  \[ h : s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{u-1}} s_u \xrightarrow{a_u} \ldots \]

• Define:
  – \( H^A \): set of all histories which end with an action
  – \( H^S \): histories which end in environment states

• A state transformer function \( \tau : H^A \rightarrow \phi(S) \) represents the effect an agent has on an environment.

• So we may represent environment dynamics by a triple:
  \[ Env = \langle S, s_0, \tau \rangle \]

• And similarly, agent dynamics as
  \[ Ag : H^S \rightarrow A \]

The architecture described in this slide and in (Wooldridge, 2002) appears to be an attempt to address some limitations of the formalisation described in the Abstract Architecture Notes. In that model, action and env — describing respectively the agent’s choice of action given an environment, and possible changes in the environment given an action — are regarded as “primitives” (in the sense that these are the basic concepts of which the other concepts in the architecture are derived). As the framework strives to be general enough to describe a large variety of agent systems, no assumptions are made about causal connections between actions and changes in the environment. Furthermore, although the history of environmental changes is assumed to be available to the agent as it chooses an action (recall the function’s signature \( \text{action} : S^* \rightarrow A \)), no specific structure exists which encodes action history.

The new model proposed here builds on efforts originating in AI (Genesereth and Nilsson, 1987, ch 14) and theoretical computer science (Fagin et al., 1995,
p 154) and apparently attempts to provide an account of both action and environment history. The primitive used for describing an agent’s behaviour is no longer an action function, as defined above, but a *history* (or *run*, as in (Fagin et al., 1995)):

\[ h : s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{u-1}} s_u \xrightarrow{a_u} \ldots \]

If one defines:
- \( H^A \): the set of all histories which end with an *action*
- \( H^S \): the histories which end in *environment states*

Then a *state transformer* function \( \tau : H^A \rightarrow \wp(S) \) represents the way the environment changes as actions are performed. (Or would it be the effect an agent has on its environment? Note that the issue of causality is never satisfactorily addressed), and the “environment dynamics” can be described as a triple:

\[ Env = < S, s_0, \tau > \]  

(1)

and the action function redefined as \( \text{action} : H^S \rightarrow A \).

Although this approach seems more adequate, (Wooldridge, 2002) soon abandons it in favour of a variant the first one when *perception* is incorporated to the framework (i.e. the action function becomes \( \text{action} : P^* \rightarrow A \)).

Utility functions

- Problem: how to “tell agents what to do”? (when exhaustive specification is impractical)

- Decision theory (see (Russell and Norvig, 1995, ch. 16)):
  - associate *utilities* (a performance measure) to states:
    \[ u : S \rightarrow \mathbb{R} \]
  - Or, better yet, to *histories*:
    \[ u : H \rightarrow \mathbb{R} \]

Example: The Tileworld

- The *utility* of a course of action can be given by:
  \[ u(h) = \frac{\text{number of holes filled in } h}{\text{number of holes that appeared in } h} \]

- When the utility function has an upper bound (as above) then we can speak of *optimal agents*. 
Optimal agents

• Let $P(h|Ag, Env)$ denote the probability that $h$ occurs when agent $Ag$ is placed in $Env$.

• Clearly
  \[ \sum_{h \in H} P(h|Ag, Env) = 1 \]  

• An optimal agent $Ag_{opt}$ in an environment $Env$ will maximise expected utility:
  \[ Ag_{opt} = \arg \max_{Ag} \sum_{h \in H} u(h)P(h|Ag, Env) \]  

From abstract to concrete architectures

• Moving from abstract to concrete architectures is a matter of further specifying action (e.g. by means of algorithmic description) and choosing an underlying form of representation.

• Different ways of specifying the action function and representing knowledge:
  – Logic-based: decision function implemented as a theorem prover (plus control layer)
  – Reactive: (hierarchical) condition → action rules
  – BDI: manipulation of data structures representing Beliefs, Desires and Intentions
  – Layered: combination of logic-based (or BDI) and reactive decision strategies

Logic-based architectures

• AKA Deductive Architectures

• Background: symbolic AI
  – Knowledge representation by means of logical formulae
  – “Syntactic” symbol manipulation
  – Specification in logic ⇒ executable specification

• “Ingredients”:
  – Internal states: sets of (say, first-order logic) formulae
    * $\Delta = \{ temp(roomA, 20), heater(on), ... \}$
  – Environment state and perception,
  – Internal state seen as a set of beliefs
  – Closure under logical implication ($\Rightarrow$):
    * $\text{closure}(\Delta, \Rightarrow) = \{ \varphi | \varphi \in \Delta \lor \exists \psi. \psi \in \Delta \land \psi \Rightarrow \varphi \}$
  – (is this a reasonable model of an agent’s beliefs?)
About whether logical closure (say, first-order logic) corresponds to a reasonable model of an agent’s beliefs, the answer is likely no (at if we are talking about human-like agents). One could, for instance, know all of axioms Peano’s axioms for natural numbers and still not know whether Goldbach’s conjecture (that all even numbers greater than 2 is the sum of two primes).

Representing deductive agents

- We will use the following objects:
  - \( L \): a set of sentences of a logical system
    * As defined, for instance, by the usual wellformedness rules for first-order logic
  - \( D = \varphi(L) \): the set of databases of \( L \)
  - \( \Delta_0, ..., \Delta_n \in D \): the agent’s internal states (or beliefs)
  - \( \models \rho \): a deduction relation described by the deduction rules \( \rho \) chosen for \( L \): We write \( \Delta \models \rho \varphi \) if \( \varphi \in \text{closure}(\Delta, \rho) \)

Describing the architecture

- A logic-based architecture is described by the following structure:

\[
\text{Arch}_L = \langle L, A, P, D, action, env, see, next \rangle
\]

- The update function consists of additions and removals of facts from the current database of internal states:
  - \( \text{next} : D \times P \rightarrow D \)
    * \( \text{old} \): removal of “old” facts
    * \( \text{new} \): addition of new facts (brought about by \( \text{action} \))

Pseudo-code for \( \text{action} \)

```
1. function action(\( \Delta : D \) : A
2.   begin
3.     for each \( a \in A \) do
4.       if \( \Delta \models \rho do(a) \) then
5.         return \( a \)
6.     end if
7.   end for
8. end function
```

4
Environment and Belief states

- The environment change function, $env$, remains as before.
- The belief database update function could be further specified as follows

$$next(\Delta, p) = (\Delta \setminus old(\Delta)) \cup new(\Delta, p)$$  \hspace{1cm} (5)

where $old(\Delta)$ represent beliefs no longer held (as a consequence of action), and $new(\Delta, p)$ new beliefs that follow from facts perceived about the new environmental conditions.

Example: The Vacuum world

(Russell and Norvig, 1995, Weiss, 1999)

Describing The Vacuum world

- Environment
  - perceptual input: dirt, null
  - directions: (facing) north, south, east, west
  - position: coordinate pairs $(x, y)$
- Actions
  - move_forward, turn_90_left, clean
- Perception:

$$P = \{\{dirt(x, y), in(x, y), facing(d), \ldots\}, \ldots\}$$
Deduction rules

- Part of the decision function
- Format (for this example):
  \[ P(...) \Rightarrow Q(...) \]
- PROLOG fans may think of these rules as **Horn Clauses**.
- Examples:
  - \( in(x,y) \land dirt(x,y) \Rightarrow do(clean) \)
  - \( in(x,2) \land \neg \text{dirt}(x,y) \land \text{facing(north)} \Rightarrow do(turn) \)
  - \( in(0,0) \land \neg \text{dirt}(x,y) \land \text{facing(north)} \Rightarrow do(forward) \)
  - ...

Updating the internal state database

- \( \text{next}(\Delta, p) = (\Delta \setminus \text{old}(\Delta)) \cup \text{new}(\Delta, p) \)
- \( \text{old}(\Delta) = \{ (P(t_1, ..., t_n) | P \in \{ \text{in, dirt, facing} \} \land P(t_1, ..., t_n) \in \Delta \} \)
- \( \text{new}(\Delta, p) : \)
  - update agent’s position,
  - update agent’s orientation,
  - etc

The “Wumpus World”

- See (Russell and Norvig, 2003, section 7.2)
- BTW, Ch 7 is available online (last accessed Oct 2012), at [http://aima.cs.berkeley.edu/newchap07.pdf](http://aima.cs.berkeley.edu/newchap07.pdf)

Shortcomings of logic-based agents

- Expressivity issues: problems encoding percepts (e.g. visual data) etc
- *Calculative rationality* in dynamic environments
- Decidability issues
- Semantic elegance vs. performance:
  - loss of “executable specification”
  - weakening the system vs. temporal specification
- etc
Existing (??) logic-based systems

- MetameM, Concurrent MetameM: specifications in temporal logic, model-checking as inference engine (Fisher, 1994)
- CONGOLOG: Situation calculus
- Situated automata: compiled logical specifications (Kaelbling and Rosenschein, 1990)
- AgentSpeak, ...
- (see (Weiss, 1999) or (Wooldridge, 2002) for more details)

BDI Agents

- Implement a combination of:
  - deductive reasoning (deliberation) and
  - planning (means-ends reasoning)

Planning

- Planning formalisms describe actions in terms of (sets of) preconditions \((P_a)\), delete lists \((D_a)\) and add lists \((A_a)\):
  \[< P_a, D_a, A_a >\]
- E.g. Action encoded in STRIPS (for the “block’s world” example):

  Stack(x,y):
  
  pre: clear(y), holding(x)
  del: clear(y), holding(x)
  add: armEmpty, on(x,y)

Planning problems

- A planning problem \(<\Delta, O, \gamma >\) is determined by:
  - the agent’s beliefs about the initial environment \((\text{a set } \Delta)\)
  - a set of operator descriptors corresponding to the actions available to the agent:
    \[O = \{< P_a, D_a, A_a > | a \in A\}\]
  - a set of formulae representing the goal/intention to be achieved (say, \(\gamma\))
- A plan \(\pi = < a_1, ..., a_n >\) determines a sequence \(\Delta_0, ..., \Delta_{n+1}\) where \(\Delta_0 = \Delta\) and \(\Delta_i = (\Delta_{i-1} \setminus D_{a_i}) \cup A_{a_i}\), for \(1 \leq i \leq n\).
Suggestions

• Investigate the use of BDI systems and agents in games.

• See, for instance, (Norling and Sonenberg, 2004) which describe the implementation of interactive BDI characters for Quake 2.

• And (Wooldridge, 2002, ch. 4), for some background.

• There are a number of BDI-based agent platforms around. ‘Jason’, for instance, seems interesting:

  http://jason.sourceforge.net/

References


