An Introduction to Multiagent Systems

LECTURE 3: DEDUCTION REASONING AGENTS

http://www.csc.liv.ac.uk/~mjw/pubs/tmas/
Introduce the idea of an agent as a computer system capable of flexible autonomous action.

Briefly discuss the issues one needs to address in order to build agent-based systems.

Three types of agent architecture:
- Hybrid.
- Reactive.
- Symbolic/logical.

An Introduction to Multiagent Systems

Lecture 3

http://www.csc.liv.ac.uk/~mjw/pubs/imas/
Lecture 3: An Introduction to Multiagent Systems

W. We want to build agents, that enjoy the properties of autonomy, reactiveness, pro-activeness, and socialability that we talked about earlier.

This is the area of agent architectures.

Maes defines an agent architecture as:

A general methodology for designing particular modular decompositions for particular tasks.

Kaelbling considers an agent architecture to be:

A particular methodology for building [agents]. It specifies how 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.

Kaelbling considers an agent architecture to be:

A specific collection of software (or hardware) modules, typically designed by boxes with arrows indicating the data and control flow among the modules. A more abstract view of an architecture is as a particular methodology for building [agents]. It specifies how 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.

Maes defines an agent architecture as:

A particular methodology for building [agents]. It specifies how 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.

W. We want to build agents, that enjoy the properties of autonomy, reactiveness, pro-activeness, and socialability that we talked about earlier.

An Introduction to Multiagent Systems

http://www.csc.liv.ac.uk/~mjw/msgs/taams/
Originally (1956-1985), pretty much all agents designed within AI were symbolic reasoning agents. Its purest expression proposed that agents use explicit logical reasoning in order to decide what to do. Problems with symbolic reasoning led to a reaction against this: reactive architectures — 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.

- Originally (1956-1985), pretty much all agents designed within AI were symbolic reasoning agents.
- Reactive agents movement, 1985-present.
- Hybrid architectures, which attempt to combine the best of reasoning and reactive architectures.
- From 1990-present, a number of alternatives proposed.
via symbolic reasoning. 
- makes decisions (for example about what actions to perform) 
World:
- contains an explicitly represented, symbolic model of the
That:
We define a deliberative agent or agent architecture to be one
This paradigm is known as Symbolic AI.
\[\text{Symbolic Reasoning} \]

2 Symbolic Reasoning Agents
If we aim to build an agent in this way, there are two key problems to be solved:

1. **The transduction problem:**
   - Automatic planning, automated reasoning, knowledge representation, automatic reasoning, etc.
   - Translating the real world into an accurate, adequate symbolic description, in time for that description to be useful.

2. **The representation/reasoning problem:**
   - Knowledge, speech understanding, learning, vision, speech, world entities and processes, and how to get agents to reason with this information in time for the results to be useful.
   - Symbolic description, in time for that description to be useful.
Most researchers accept that neither problem is anywhere near solved.

Because of these problems, some researchers have looked to alternative techniques for building agents; we look at these later.

Underlying problem lies with the complexity of symbol manipulation algorithms in general: many (most) search-based algorithms of interest are highly intractable.

Most researchers accept that neither problem is anywhere near solved.
How can an agent decide what to do using theorem proving?

Let:

- \( \phi \) be a theory (typically a set of rules);
- \( \Delta \) be the set of actions the agent can perform;
- \( \Delta \) be a logical database that describes the current state of the world;
- \( d \) be this theory (typically a set of rules).

\( \phi \models d \Delta \) means that \( \phi \) can be proved from \( \Delta \) using logic.

- How can an agent decide what to do using theorem proving?
/* try to find an action explicitly prescribed */
for each \( a \in A_c \) do
  if \( \Delta \models \rho_D(o(a)) \) then
    if \( \Delta \models \neg\rho_D(o(a)) \) then
      return \( o(a) \)
    else
      return \( a \)
  end-if
end-for

/* try to find an action not excluded */
for each \( a \in A_c \) do
  if \( \Delta \models \rho_D(o(a)) \) then
    if \( \Delta \models \neg\rho_D(o(a)) \) then
      return \( o(a) \)
    else
      return \( a \)
  end-if
end-for

return null /* no action found */
Goal is for the robot to clear up all dirt.

An example: The Vacuum World.
An Introduction to Multiagent Systems

Lecture 3

Use 3 domain predicates in this exercise:

- \( \text{Dirt}(x, y) \): There is dirt at \( x, y \)
- \( \text{In}(x, y) \): Agent is at \( x, y \)
- \( \text{Facing}(d, x) \): Agent is facing direction \( d \)

Possible actions:

- \( \text{turn} \) means “turn right.”

\[ \{ \text{turn, forward}, \text{such} \} = \text{Ac} \]

NB: \( \text{turn} \) means “turn right.”

An Introduction to Multiagent Systems

Lecture 3
Using these rules (+ other obvious ones), starting at (0, 0) the robot will clear up dirt.

\[
\begin{align*}
\text{In} \ 0, 0 & \quad \text{forward} \quad \leftarrow \ (\text{In} \ 0, 2) \lor (\text{Facing} \ \text{east}) \\
\text{In} \ 0, 0 & \quad \text{turn} \quad \leftarrow \ (\text{In} \ 0, 2) \lor (\text{Facing} \ \text{north}) \lor (\text{Dirt} \ 0, 2) \\
\text{In} \ 0, 0 & \quad \text{turn} \quad \leftarrow (\text{In} \ 0, 1) \lor (\text{Facing} \ \text{north}) \lor (\text{Dirt} \ 0, 1) \\
\text{In} \ 0, 0 & \quad \text{turn} \quad \leftarrow (\text{In} \ 0, 0) \lor (\text{Facing} \ \text{north}) \lor (\text{Dirt} \ 0, 0)
\end{align*}
\]

Rules for determining what to do:

... and so on!
Problems:

- How to convert video camera input to Dirt?
- Decision making assumes a static environment: calculative rationality.
- Decision making using first-order logic is undecidable.
- Weaken the logic;
- Use symbolic, non-logical representations;
- Shift the emphasis of reasoning from runtime to design time.

Typical solutions:

(NB: co-NP-complete = bad news!)

Even where we use propositional logic, decision making in the worst case means solving co-NP-complete problems.

An Introduction to Multiagent Systems

http://www.csc.liv.ac.uk/~mjw/pubs/tmas/
MuchoftheinterestinagentsfromtheAIcommunityhasarisen
fromShoham’snotionofagentorientedprogramming(AOP).

AOPa‘newprogrammingparadigm,basedonasocietalviewof
computation.’ThekeyideathatinformsAOPisthatofdirectlyprogramming
agents intermsofintentionalnotionslikebelief,commitment,
andintention.

The idea that informs AOP is that of directly programming
computation.

Let’s use the intentional stance to describe
representing the properties of complex systems.

The motivation behind such a proposal is that, as we humans

program machines, it might be useful to use the intentional stance to
describe
humans, in the same way that we use the intentional stance to
describe

Much of the interest in agents from the AI community has arisen

2.2 AGENT0 and PLACA
Shoham suggested that a complete AOP system will have 3 components:

1. A logic for specifying agents and describing their mental states;
2. An interpreted programming language for programming agents;
3. An 'agentification' process, for converting 'neutral applications' (e.g., databases) into agents.

Results only reported on first two components.

Relationship between logic and programming language is semantics.

We will skip over the logic(i), and consider the first AOP language, AGENT0. An interpreted programming language for programming agents; an 'agentification' process, for converting neutral applications; a logic for specifying agents and describing their mental states; components.
Each agent in AGENT₀ has 4 components:

AGENT₀ is implemented as an extension to LISP.

The key component, which determines how the agent acts, is the commitment rule set.

- a set of commitment rules;
- a set of initial commitments (things the agent will do); and
- a set of initial beliefs;
- a set of capabilities (things the agent can do).

AGENT₀ is implemented as an extension to LISP.
Each commitment rule contains:
- a message condition;
- a mental condition; and
- an action.

If the rule fires, then the agent becomes committed to the action.

On each ‘agent cycle’...

The mental condition is matched against the beliefs of the agent.
The message condition is matched against the messages the agent has received.

The agent becomes committed to the action.
Actions may be

- private:
  - an internally executed computation, or
  - communicative:
    - sending messages.

Messages are constrained to be one of three types:

- "requests" to commit to action;
- "unrequests" to refrain from actions;
- "informs" which pass on information.
A commitment rule:

\[
\text{COMMIT}((\text{agent}, \text{REQUEST}, \text{DO}(\text{time}, \text{action})), \text{msgcondition})
\]

\[
\text{msgcondition} = \text{B}.
\]

\[
\text{COMMIT}(\text{agent}, \text{REQUEST}, \text{DO}(\text{time}, \text{action})).
\]

\[
\text{self} \quad \text{mental condition} \quad ! ! ! ! !
\]

\[
\text{CAN} (\text{self}, \text{action} \quad \text{AND} \quad \text{now}, \text{friend agent} \quad \text{AND} \quad \text{NOT}[\text{time}, \text{CMT} (\text{self}, \text{any action})]
\]

\[
\text{self}, \text{DO}(\text{time}, \text{action})
\]
This rule may be paraphrased as follows:

- If I receive a message from agent which requests me to do action at time, and I believe that:
  - agent is currently a friend;
  - I can do the action;
  - at time, I am not committed to doing any other action,
then commit to doing action at time.
AGENT0 provides support for multiple agents to cooperate and communicate, and provides basic provision for debugging.

However, it is, however, a prototype, that was designed to illustrate some principles, rather than be a production language.

Her Planning Communicating Agents (PLACA) language was intended to address one severe drawback to AGENT0: the inability of agents to plan, and communicate requests for action via high-level goals.

A more refined implementation was developed by Thomas, for her 1993 doctoral thesis.

Agents in PLACA are programmed in much the same way as in AGENT0, in terms of mental change rules.

Her Planning Communicating Agents (PLACA) language was intended to address one severe drawback to AGENT0: the inability of agents to plan, and communicate requests for action via high-level goals.

It is, however, a prototype, that was designed to illustrate some principles, rather than be a production language.

Agents in PLACA are programmed in much the same way as in AGENT0, in terms of mental change rules.

An Introduction to Multiagent Systems
An example mental change rule:

- Inform them of your newly adopted intention.
- Adopt the intention to xerox it by 5pm, and shelving books, then don't believe that they're a VIP, or that you're supposed to be!

Paraphrased:

```
(((self?agent REQUEST(?t(xeroxed?x)))
   (AND(CAN-ACHIEVE(?txeroxed?x))
     (NOT(BEL(*now*shelving))
      (NOT(BEL(*now*(VIP?agent))))
     ((ADOPT(INTEND(5pm(xeroxed?x))))
      ((?agentself INFORM(*now*(INTEND(5pm(xeroxed?x)))))))
```

http://www.csc.liv.ac.uk/~mjw/pubs/imas/22
Concurrent MetateM is a multi-agent language in which each agent is programmed by giving it a temporal logic specification. These specifications are executed directly in order to generate the behaviour of the agent. Temporal logic is classical logic augmented by modal operators for describing how the truth of propositions changes over time.
Lecture 3

An Introduction to Multiagent Systems

For example...

- important(agents) means “it is now, and will always be true that agents are important”
- important(ConcurrentMetateM) means “sometime in the future, ConcurrentMetateM will be important”
- important(Prolog) means “sometime in the past it was true that Prolog was important”
- important(friends(us)) means “we are not friends until you apologise
  (-apologise(you)) means “tomorrow (in the next state), you apologise”
A MetateM program is a set of such rules.

This past ↔ future form can be used as execution rules.

Any arbitrary temporal logic formula can be rewritten in a logically equivalent past ↔ future form.

The root of the MetateM concept is Gabbay's separation theorem:

The instantiated future-time consequents become commitments which must subsequently be satisfied.

Execution proceeds by a process of continually matching rules against a "history", and firing those rules whose antecedents are satisfied.

MetateM is a framework for directly executing temporal logic specifications.
Execution is thus the process of iteratively generating a model for the formula made up of the program rules. The future-time parts of instantiated rules represent constraints on this model. There are algorithms for executing MetateM programs that appear to give reasonable performance. There is also a separated normal form.

An example MetateM program: the resource controller...

\begin{align*}
\forall x,y \forall \text{give}(x) \vee \text{give}(y) \land & \text{ask}(x) \Leftrightarrow \text{give}(x) \land \text{give}(y) \\
\forall x \text{give}(x) \Leftrightarrow & \text{give}(x) \\
\forall x \text{ask}(x) \Leftrightarrow & \emptyset
\end{align*}

An example MetateM program: the resource controller

Lecture 3

http://www.csc.liv.ac.uk/~mjw/pubs/tmass/
ConcurrentMetaM provides an operational framework through which societies of MetaM processes can operate and communicate. It is based on a new model for concurrency in executable logics: the notion of executing a logical specification to generate individual agent behaviour.

A ConcurrentMetaM system contains a number of agents:

- a MetaM program.
- an interface.
- a name.

(Objects), each object has 3 attributes:

- a name;
- an interface;
- a MetaM program.

A ConcurrentMetaM system contains a number of agents.
An object's interface contains two sets:

- environment predicates—these correspond to a component predicate, it accepts if an object satisfies a commitment corresponding to a component predicate, it broadcasts if an agent receives a message headed by an environment predicate.

For example, a 'stack' object's interface:

- may send:

  - component predicates—corresponding to messages the object will accept;

  - environment predicates—these correspond to messages the object contains two sets:
To illustrate the language Concurrent MetaM in more detail, here are some example programs.

**Snow White**

```plaintext
Snow-White(ask)[give]:

\( (\forall x) (\exists y) \text{give}(x) \land \text{give}(y) \)
\( \text{ask}(x) \Rightarrow \text{give}(x) \)
\( \text{give}(y) \Rightarrow (x = y) \)
```

Here is Snow White, written in Concurrent MetaM:

- Snow White has some sweets (resources), which she will give to the Dwarves (resource consumers).
- She will only give to one dwarf at a time.
- She will always eventually give to a dwarf that asks.
- She will only give to one dwarf at a time.

Lecture 3
The dwarf ‘eager’ asks for a sweet initially, and then whenever he has just received one, asks again.

The dwarf ‘greedy’ asks for a sweet initially, and then whenever he

Some dwarves are even less polite: ‘greedy’ just asks every time.

Lecture 3
An Introduction to Multiagent Systems

http://www.csc.liv.ac.uk/~mjw/pubs/tmas/
Fortunately, some have better manners; courteous only asks when 'eager' and 'greedy' have eaten. And finally, 'shy' will only ask for a sweet when no-one else has just asked.

\begin{align*}
\text{ask(courteous)} & \Leftarrow \neg \text{ask(shy)} \cup \neg \text{give(greedy)} \cup \neg \text{give(eager)} \\
\text{start} & \Leftarrow \neg \text{give}([]) \\
\text{shy(give)} & \Leftarrow \text{ask}([])
\end{align*}
Summary:

- currently prototype only; full version on the way!
- not be used in current state to implement "full" system.
- but unfortunately, lacks many desirable features — could
  very nice underlying theory.
- (another) experimental language;

http://www.csc.liv.ac.uk/~mjw/pubs/tmas/
2.4 Planning agents

Since the early 1970s, the AI planning community has been closely concerned with the design of artificial agents.

Planning is essentially automatic programming: the design of a course of action that will achieve some desired goal.

Within the symbolic AI community, it has long been assumed that some form of AI planning system will be a central component of any artificial agent.

Building largely on the early work of Fikes & Nilsson, many planning algorithms have been proposed, and the theory of planning has been well-developed.

But in the mid 1980s, Chapman established some theoretical results which indicate that AI planners will ultimately turn out to be unusable in any time-constrained system.