Agent-Based Systems

Michael Rovatsos
mrovatso@inf.ed.ac.uk

Lecture 3 – Deductive Reasoning Agents

Where are we?

Last time . . .

- Talked about abstract agent architectures
- Agents with and without state
- Goals and utilities
- Task environments

Today . . .

- **Deductive Reasoning Agents**

Deductive reasoning agents

- After abstract agent architectures we have to make things more concrete ➡ we take viewpoint of “symbolic AI” as a starting point
- Main assumptions:
  - Agents use symbolic representations of the world around them
  - They can reason about the world by syntactically manipulating symbols
  - Sufficient to achieve intelligent behaviour (“symbol system hypothesis”)
- Deductive reasoning = specific kind of symbolic approach where representations are **logical formulae** and syntactic manipulation used is **logical deduction (theorem proving)**
- Core issues: **transduction problem**, **representation/reasoning problem**

Agents as theorem provers

- Simple model of “deliberate” agents: internal state is a database of first-order logic formulae
- This information corresponds to the “belief” of the agent (may be erroneous, out of date, etc.)
- L set of sentences of first-order logic, $D = \wp(L)$ set of all L-databases (=set of internal agent states)
- Write $\Delta \vdash_\rho \varphi$ if φ can be proved from DB $\Delta \in D$ using (only) deduction rules ρ
- Modify our abstract architecture specification:

$$\begin{aligned} see &: E \rightarrow Per \\ action &: D \rightarrow Ac \\ next &: D \times Per \rightarrow D \end{aligned}$$

Agents as theorem provers

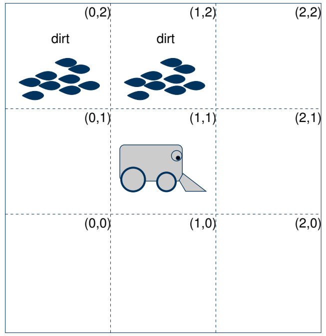
- Assume special predicate $Do(\alpha)$ for action description α
- If $Do(\alpha)$ can be derived, α is the best action to perform
- Control loop:

```
Function: Action Selection as Theorem Proving
function action(Δ : D) returns an action Ac
    for each α ∈ Ac do
        if Δ ⊢_ρ Do(α) then
            return α
    for each α ∈ Ac do
        if Δ ⊬_ρ ¬Do(α) then
            return α
    return null
```

- If no "good" action is found, agent searches for consistent actions instead (that are not explicitly forbidden)
- Do you notice any problems here?

Example: the vacuum world

- A small robot to help with housework
  - Perception: dirt sensor, orientation (north, south, east, west)
  - Actions: suck up dirt, step forward, turn right by 90 degrees
  - Starting point $(0, 0)$, robot cannot exit room



- Goal: traverse the room continually, search for and remove dirt

Example: the vacuum world

- Formulate this problem in logical terms:
- Percept is *dirt* or *null*, actions *forward*, *suck* or *turn*
- Domain predicates $In(x, y)$, $Dirt(x, y)$, $Facing(d)$
- *next* function must update internal (belief) state of agent correctly
  - $old(\Delta) := \{P(t_1 \dots t_n) | P \in \{In, Dirt, Facing\} \wedge P(t_1 \dots t_n) \in \Delta\}$
  - Assume $new : D \times Per \rightarrow D$ adds new predicates to database (what does this function look like?)
  - Then, $next(\Delta, p) = (\Delta \backslash old(\Delta)) \cup new(\Delta, p)$
- Agent behaviour specified by (hardwired) rules, e.g.

$$In(x, y) \wedge Dirt(x, y) \Rightarrow Do(suck)$$
$$In(0, 0) \wedge Facing(north) \wedge \neg Dirt(0, 0) \Rightarrow Do(forward)$$
$$In(0, 1) \wedge Facing(north) \wedge \neg Dirt(0, 1) \Rightarrow Do(forward)$$
$$In(0, 2) \wedge Facing(north) \wedge \neg Dirt(0, 2) \Rightarrow Do(turn)$$
$$In(0, 2) \wedge Facing(east) \Rightarrow Do(forward)$$

Critique of the DR approach

- How useful is this kind of agent design in practice?
- Naive implementation of this certainly won't work!
- What if world changes since optimal action was calculated?
  ➡ notion of **calculative rationality** (decision of system was optimal when decision making began)
- In case of first-order logic, not even termination is guaranteed . . . (let alone real-time behaviour)
- Also, formalisation of real-world environments (esp. sensor input) often counter-intuitive or cumbersome
- Clear advantage: elegant semantics, declarative flavour, simplicity

Agent-oriented programming

- Based on Shoham's (1993) idea of bringing societal view into agent programming (AGENT0 programming language)
- Programming agents in terms of **mentalistic** notions (beliefs, desires, intentions)
- Agent specified in terms of
  - set of capabilities
  - set of initial beliefs
  - set of initial **commitments**
  - set of **commitment rules**
- Key component: commitment rules, composed of message condition, mental condition and action (private or communicative)
- Rule matching used to determine whether rule should fire
- Messages types: requests, unrequests (change commitments), inform messages (change beliefs)

Agent-oriented programming

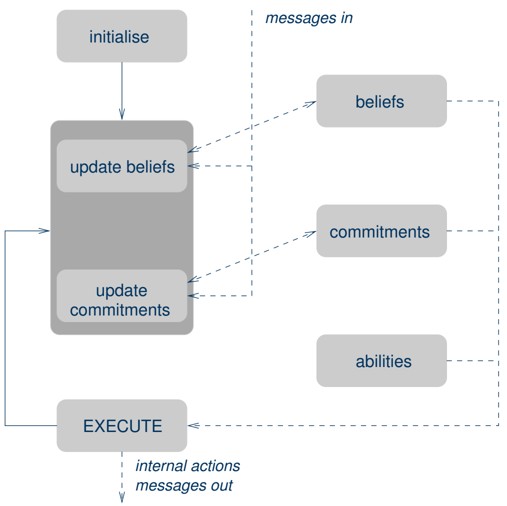
- Suppose we want to describe commitment rule

  "**If** I receive a message from *agent* requesting me to do *action* at *time* and I believe that (a) *agent* is a friend, (b) I can do the action and (c) at *time* I am not committed to doing any other action **then** commit to *action* at *time*"
- This is what this looks like in AGENT0:

```
COMMIT(agent,REQUEST,DO(time,action)
(B,[now,Friend agent] AND CAN(self,action) AND NOT
[time,CMT(self,anyaction)]),
self, DO(time,action))
```

- Top-level control loop used to describe AGENT0 operation:
  - Read all messages, update beliefs and commitments
  - Execute all commitments with satisfied capability condition
  - Loop.

Agent-oriented programming



Concurrent MetateM

- Based on direct execution of logical formulae
- Concurrently executing agents communicate via asynchronous broadcast message passing
- Agents programmed by **temporal logic** specification
- Two-part agent specification
  - **interface** defines how agent interacts with other agents
  - **computational engine** which defines how agent will act
- Agent interface consists of
  - unique agent identifier
  - "environment propositions", i.e. messages accepted by the agent
  - "component propositions", i.e. messages agent will send
- Example: *stack*(*pop*, *push*)[*popped*, *full*]

Concurrent MetateM

- Computational engine based on MetateM, based on program rules:

  antecedent about past $\Rightarrow$ *consequent about present and future*

- "Declarative past and imperative future" paradigm
- Agents are trying to make present and future true given past
  - Collect constraints with old commitments
  - These taken together form current constraints
  - Next state is constructed by trying to fulfil these
  - Disjunctive formula ➡ choices
  - Unsatisfied commitments are carried over to the next cycle

Propositional MetateM logic

- Propositional logic with (lots of) temporal operators

$\bigcirc\varphi$	φ is true tomorrow
$\circledcirc\varphi$	φ was true yesterday
$\Diamond\varphi$	φ now or at some point in the future
$\Box\varphi$	φ now and at all points in the future
$\blacklozenge\varphi$	φ was true sometimes in the past
$\blacksquare\varphi$	φ was always true in the past
$\varphi\,\mathcal{U}\,\psi$	ψ some time in the future φ until then
$\varphi\,\mathcal{S}\,\psi$	ψ some time in the past, φ since then (but not now)
$\varphi\,\mathcal{W}\,\psi$	ψ was true unless φ was true in the past
$\varphi\,\mathcal{Z}\,\psi$	like "$\mathcal{S}$" but φ may have never become true

- Beginning of time: special nullary operator (*start*) satisfied only at the beginning

Agent execution

- Some examples:
  - $\Box important(agents)$: "now and for all times agents are important"
  - $\Diamond important(agents)$: "agents will be important at some point"
  - $\neg friends(us)\,\mathcal{U}\,apologise(you)$: "not friends until you apologise"
  - $\bigcirc apologise(you)$: "you will apologise tomorrow"
- Agent execution: attempt to match past-time antecedents of rules against **history**, executing consequents of rules that fire
- More precisely:
  1. Update history with received messages (environment propositions)
  2. Check which rules fire by comparing antecedents with history
  3. **Jointly** execute fired rule consequents together with commitments carried over from previous cycles
  4. Goto 1.

Example

- Specification of an example system:

$$rp(ask1, ask2)[give1, give2]:$$
$$\circledcirc ask1 \Rightarrow \Diamond give1$$
$$\circledcirc ask2 \Rightarrow \Diamond give2$$
$$start \Rightarrow \Box\neg(give1 \wedge give2)$$
$$rc1(give1)[ask1]:$$
$$start \Rightarrow ask1$$
$$\circledcirc ask1 \Rightarrow ask1$$
$$rc2(ask1, give2)[ask2]:$$
$$\circledcirc(ask1 \wedge \neg ask2) \Rightarrow ask2$$

- What does it do?

Example

- *rp* resource producer, cannot *give* to both agents at a time, but will give eventually to any agent that asks
- *rc*1/*rc*2 are resource consumers:
  - *rc*1 will ask in every cycle
  - *rc*2 will always ask if it has not asked previously and *rc*1 has asked
- Example run:

time	*rp*	*rc*1	*rc*2
0		*ask*1	
1	*ask*1	*ask*1	*ask*2
2	*ask*1,*ask*2,*give*1	*ask*1	
3	*ask*1,*give*2	*ask*1,*give*1	*ask*2
4	*ask*1,*ask*2,*give*1	*ask*1	*give*2
5	...	...	...

Summary

- Deductive reasoning agents
- Working with pure logic specifications of agent behaviour
- General architecture, vacuum cleaner example
- Critique: elegant, but complexity and practicability issues
- Agent-oriented programming: first approach to use mentalistic concepts in programming (but not a true programming language)
- Concurrent MetateM: powerful and expressive but somewhat specific
- Next time: **Practical Reasoning Agents**