Multiagent Systems

2. Deductive Reasoning Agents

B. Nebel, C. Becker-Asano, S. Wölfl

Albert-Ludwigs-Universität Freiburg

May 7, 2014

Multiagent Systems

May 7, 2014 — 2. Deductive Reasoning Agents

2.1 Introduction

- Agent Architectures
- Symbolic reasoning agents

What are agent architectures?

An **agent architecture** is a **software design** for an agent.

The last lecture introduced the **top-level decomposition** into:

perception – state – decision – action

An agent architecture defines:

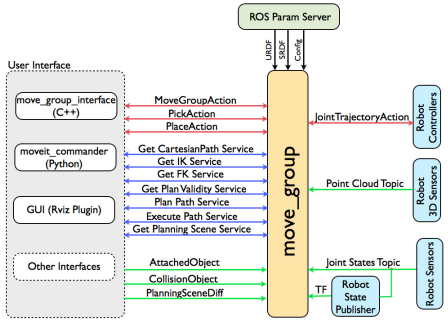
- Key data structures
- operations on data structures
- control flow between operations

Agent Architectures

"[A] particular methodology for building [agents]. It specifies how ... the agent can be decomposed into the construction of a set of component modules and how these modules should be made to interact. 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. An architecture encompasses techniques and algorithms that support this methodology." (Pattie Maes, 1991)

"[A] specific collection of software (or hardware) modules, typically designated by boxes with arrows indicating the data and control flow among the modules. A more abstract view of an architecture is as a general methodology for designing particular modular decomposition for particular tasks."
(Leslie Kaelbing, 1991)

Example: "MoveIt" component for ROS



"[The] high-level system architecture for the primary node provided by MoveIt! called move_group, pulling all the individual components together to provide a set of ROS actions and services for users to use."
(`http://moveit.ros.org/documentation/concepts/`)

Types of Agents

- 1956 – present: **Symbolic Reasoning Agents**
  Its purest expression, proposes that agents use explicit logical reasoning in order to decide what to do
- 1985 – present: **Reactive Agents**
  Problems with symbolic reasoning led to a reaction against this – led to the reactive agents movement
- 1990 – present: **Hybrid Agents**
  Hybrid architectures attempt to combine the best of symbolic and reactive architectures

Symbolic Reasoning Agents

The classical approach to building agents:

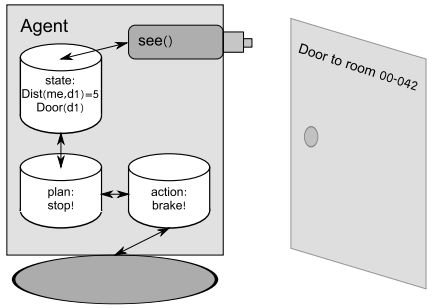
- Agents as a particular type of **knowledge-based system**
- Make use of associated methodologies
- Paradigm known as **symbolic AI**

Definition 16: Deliberative Agent (Architecture)

A deliberative agent or agent architecture is one that:

- contains an explicitly represented, symbolic model of the world, and
- makes decisions (e.g. about actions to perform) via symbolic reasoning.

Representing the environment symbolically



The **transduction problem**:

- how to translate the real world into an accurate, adequate symbolic description
- in time for that description to be useful ⇒ vision, speech understanding, etc.

Problems with Symbolic Approaches

The **representation/reasoning problem**:

- how to **symbolically represent** information about complex real-world entities and processes
- how to let agents reason with this information **in time** for the results to be useful ⇒ knowledge representation, automated reasoning, planning

In general:

- **Real-world problems** (apart from games like chess) are very hard to be solved this way
- Underlying problem is the complexity of symbol manipulation algorithms in general, e.g. **intractability** of of search-based symbol manipulation algorithms
- These problems let to **alternative approaches** discussed later...

2.2 Deductive Reasoning Agents

- Agents as theorem provers
- Agent-oriented programming (AOP)

Deductive Reasoning Agents

Main assumptions:

- Agents use symbolic representations of the world around them
- They reason about the world by syntactically manipulating symbols
- Assumed sufficient to achieve **intelligent behavior** according to the "symbol system hypothesis"

Deductive reasoning ⇒ specific kind of symbolic approach where representations are **logical formulae** and syntactic manipulation is achieved by **logical deduction (theorem proving)**

Agents as theorem provers – background

Simple model of "deliberate" agents:

- Internal state is a database of first-order logic formulae
- Corresponds to the "beliefs" of the agent (may be erroneous, out of date, etc.)
- Let L be the set of sentences of first-order logic, $D = \mathcal{P}(L)$ be the set of all L-databases (i.e., set of internal agent states)
- Write $\Delta \vdash_\rho \psi$ if ψ can be proved from DB $\Delta \in D$ using (only) deduction rules ρ

Modify abstract agent architecture specification:

$$\text{see: } E \to Per \quad (1)$$

$$\text{action: } D \to Ac \quad (2)$$

$$\text{next: } D \times Per \to D \quad (3)$$

Agents as theorem provers – background

Action selection as theorem proving

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

Control loop:

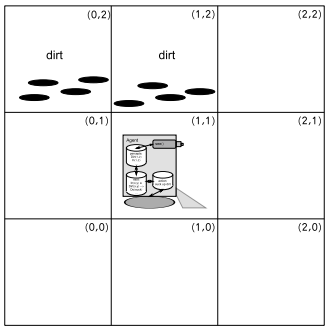
```
function action(Δ : D): returns an action α ∈ Ac
    foreach α ∈ Ac do
        if Δ ⊢_ρ Do(α) then
            return α;
    end
    foreach α ∈ Ac do
        if Δ ⊬_ρ ¬Do(α) then
            return α;
    end
    return null;
```

If no "good" action ⇒ search for **consistent** action instead.

Example: the vacuum world

A small robot to help with housework:

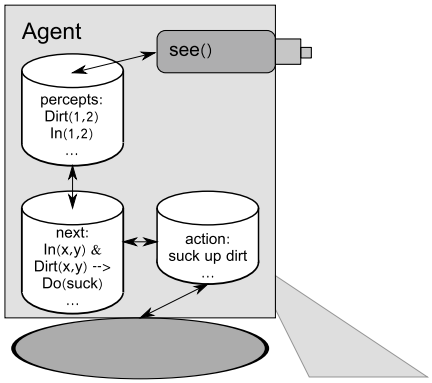
- Perceptions: dirt sensor, orientation (N, S, E, W)
- Actions: suck up dirt, step forward, turn right (90°)
- Starting point (0, 0), robot cannot exit the room



Goal: traverse room continually, search for dirt and remove it

Example: the agent for the vacuum world

A sketch of the agent:



Example: Logical formulation

Formulate this problem in logical terms:

- Percept is either **dirt** or **null**
- Actions are **forward**, **suck**, and **turn**
- Domain predicates are **In(x,y)**, **Dirt(x,y)**, **Facing(d)**

next() function must update internal (belief) state of agent:

$$old(\Delta) := \{P(t_1 \dots t_n) | P \in \{In, Dirt, Facing\} \wedge P(t_1 \dots t_n) \in \Delta\}$$

Assume $new : D \times Per \rightarrow D$ adds new predicates to database, then $next(\Delta, p) = (\Delta / old(\Delta)) \cup new(\Delta, p)$.
Agent behavior specified by **hardwired** rules, e.g.:

$$\begin{aligned} In(x,y) \wedge Dirt(x,y) &\Rightarrow Do(suck) \\ In(0,0) \wedge Facing(N) \wedge \neg Dirt(0,0) &\Rightarrow Do(foreward) \\ In(0,1) \wedge Facing(N) \wedge \neg Dirt(0,1) &\Rightarrow Do(foreward) \\ In(0,2) \wedge Facing(N) \wedge \neg Dirt(0,2) &\Rightarrow Do(turn) \\ In(0,2) \wedge Facing(E) &\Rightarrow Do(forward) \end{aligned}$$

Critique of the symbolic approach

How useful is this kind of agent design in practice?

- Naive implementation certainly won't work!
- What if world changes after optimal action was chosen?
  $\Rightarrow$ notion of **calculative rationality**, i.e. decision of system optimal, when decision making began
- In case of first-order logic, not even termination can be guaranteed (undecidability) ...let alone real time behavior
- Formalization of real-world environments often difficult and counter-intuitive
- Clear advantage: elegant semantics, declarative flavor, simplicity

Agent oriented programming

Based on Shoham's (1993) idea of bringing societal view into agent programming (AGENT0 programming language). Programming agents using 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 determines whether rule should fire
- message types are **requests**, **unrequests** (change commitments), and **inform messages** (change beliefs)

AOP – commitment rules in AGENT0

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 can be expressed in AGENT0 like so:

```
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.

Concurrent MetateM

Features of **Concurrent MetateM**:

- Based on direct execution of logical formulae
- Concurrently executing agents communicate via asynchronous broadcast message passing
- Two-part agent specification:
  - **interface** defines how agent interacts with other agents
  - **computational engine** defines how agent will act
- Agent interface consists of:
  - unique agent identifier
  - "environment propositions", i.e. a set of symbols specifying which messages the agent accepts
  - "component propositions", i.e. a set of symbols specifying messages the agent will send
- Example: interface definition of "stack"
  $\Rightarrow stack(pop, push)[popped, full]$

Concurrent MetateM – program rules

Computational engine of `Concurrent MetateM` is based on `MetateM`, which is based on **program rules**, which are **temporal logic formaulae** of the form:

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

"Declarative past and imperative future" paradigm (Gabbay, 1989)

Agents try to make present and future true, given the past:

- Collect constraints with old commitments
- These taken together form current constraints
- Next state is constructed by trying to fulfil these
- Disjunctive formula $\Rightarrow$ choices
- Unsatisfied commitments are carried over to the next cycle

Propositional MetateM logic

Propositional logic with 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"
- $\neg friends(us)\ \mathcal{U}\ apologize(you)$: "not friends until you apologize"
- $\bigcirc apologize(you)$: "you will apologize tomorrow"

Agent execution:
Attempt to match past-time antecedents of rules against **history** and execute 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. Loop.

Specification of an example system

Consider the following definition of a system:

$$
\begin{aligned}
rp(ask_1, ask_2)[give_1, give_2] :& \\
\circledcirc ask_1 &\Rightarrow \Diamond give_1 \\
\circledcirc ask_2 &\Rightarrow \Diamond give_2 \\
start &\Rightarrow \Box\neg(give_1 \wedge give_2) \\
rc_1(give_1)[ask_1] :& \\
start &\Rightarrow ask_1 \\
\circledcirc ask_1 &\Rightarrow ask_1 \\
rc_2(ask_1, give_2)[ask_2] :& \\
\circledcirc(ask_1 \wedge \neg ask_2) &\Rightarrow ask_2
\end{aligned}
$$

What does it do?

Example run

rp is [r]esource [p]roducer, cannot *give* to both agents simultaneously, but will give eventually to any agent that asks.
rc_1 and rc_2 are resource consumers:

- rc_1 will ask in every cycle
- rc_2 only asks 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	...	...	...

2.3 Summary

- Thanks

Summary

- Agent architectures / MoveIt (ROS)
- Symbolic Reasoning Agents
- Agents as theorem provers
- General architecture, vacuum world example
- Agent-oriented programming (AGENT0): first approach to use mentalistic concepts in programming (but not a true programming language)
- Concurrent MetateM & temporal logic: powerful and expressive but somewhat specific

⇒ Next time: **Practical reasoning agents**

Acknowledgments

These lecture slides are partly based on the following slides:

- Dr. Michael Rovatsos, The University of Edinburgh
  http://www.inf.ed.ac.uk/teaching/courses/abs/abs-timetable.html
- Prof. Micheal Wooldridge, University of Oxford
  http://www.cs.ox.ac.uk/people/michael.wooldridge/pubs/imas/distrib/pdf-index.html