In Control of Autonomous Decision Systems

Language Design for Cognitive Agents and Artificial Intelligence

Koen Hindriks
Delft University of Technology, The Netherlands

9-9-2015 @ DTU
Outline

• First Step: Agent Programming

• Second Step: Building on Top of KRTs

• Third Step: Towards AI Programming
First Step: Agent Programming
EMAS14 audience listed the following key concepts:

- autonomy
- rational
- goal-directedness
- interaction
- social
- reactive/events
- environment
- robustness
- decentralization
- Intentional stance
## Engineering Approaches?

<table>
<thead>
<tr>
<th>Theories of intelligent agents: How do the various components of an agent's cognitive makeup conspire to produce rational behaviour?</th>
</tr>
</thead>
</table>
| **intentions**
| time, desires, beliefs, goals
| situated automata
| logical models of agents
| executing agent specs (bounded) rationality |

<table>
<thead>
<tr>
<th>Architectures for intelligent agents: What structure should an artificial intelligent agent have?</th>
</tr>
</thead>
</table>
| deliberative architectures
| reactive architectures
| hybrid architectures |

<table>
<thead>
<tr>
<th>Languages for intelligent agents: What are the right primitives for programming an intelligent agent?</th>
</tr>
</thead>
</table>
| agent spec languages
| agent-oriented paradigm
| non-logical agent languages
| agent-based computing |
Illustrative Architecture: InteRRaP

- Layered architectures, e.g., **InteRRaP** agent model:

- **dMARS** architecture (MAS extension of PRS)
- Early work on coordination & organizations
Evolution of Programming

- Organisation-based design
  - e.g. GOAL
  - Agent-oriented programming (AOP)
- Object-oriented languages (OOP)
  - e.g. Prolog
- Knowledge-oriented languages (KR)
- Procedural languages
  - Assembler
  - Machine language

As time progresses, abstraction increases.

Koen Hindriks  In Control of Autonomous Decision Systems
Programming with Mental States

Cognitive Agent

Agent program

decision rules

Mental state

beliefs

goals

Environment

e.g.,
or

Koen Hindriks
In Control of Autonomous Decision Systems
Many Agent Programming Languages

The landscape of agent frameworks presented and introduced @ATAL. Includes operational agent languages and logical models.

In a sense this landscape defines a space of agents that can be created (and thus a corresponding mindset).
How are these APLs related?

A comparison from a high-level, conceptual point, not taking into account any practical aspects (IDE, available docs, speed, applications, etc)

Basic concepts: beliefs, action, plans, goals-to-do

AGENT-0\(^1\)
(PLACA)

AgentSpeak(L), Jason\(^2\)

Golog \(=\) 3APL\(^3\)

Main addition: Declarative goals

2APL \(\approx\) 3APL + GOAL

Java-based Cognitive Agent Languages

AF-APL, JACK (commercial), Jadex, Jazzyk

Families of Languages

Logic Programming

METATEM

Mobile Agents

CLAIM

\(^1\) mainly interesting from a historical point of view

\(^2\) from a conceptual point of view, we identify AgentSpeak(L) and Jason

\(^3\) without practical reasoning rules
Introducing \textbf{GOAL}

Blocks World Toy Example
The Blocks World

Objective: Move blocks in initial state such that result is goal state.

- Positioning of blocks on table is not relevant.
- A block can be moved only if there is no other block on top of it.
Representing the Initial State

Using the on(X,Y) predicate we can represent the initial state.

Initial belief base of agent

```prolog
beliefs{
   on(a,b).
   on(b,c).
   on(c,table).
   on(d,e).
   on(e,table).
   on(f,g).
   on(g,table).
}
```
Domain Knowledge

- Domain knowledge is added to the knowledge base.

```
tower([X]) :- on(X,table).
tower([X,Y|T]) :- on(X,Y), tower([Y|T]).
clear(X) :- block(X), not(on(Y,X)).
clear(table).
```

```
knowledge{
clear(X) :- not(on(_,X)).
clear(table).
tower([X]) :- on(X,table).
tower([X,Y|T]) :- on(X,Y), tower([Y|T]).
}
```

Static knowledge base of agent
Using the on(X,Y) predicate we can represent the goal state.

Representing the Goal State

goals{
  on(a,e),
  on(b,table),
  on(c,table),
  on(d,c),
  on(e,b),
  on(f,d),
  on(g,table).
}

Initial goal base of agent
Mental State of Agent

The knowledge, belief, and goal sections together constitute the specification of the mental state of the agent.

```
knowledge{
  clear(X) :- not(on(_,X)).
  clear(table).
  tower([X]) :- on(X,table).
  tower([X,Y|T]) :- on(X,Y), tower([Y|T]).
}

beliefs{
  on(a,b). on(b,c). on(c,table). on(d,e). on(e,table).
  on(f,g). on(g,table).
}

goals{
  on(a,e), on(b,table), on(c,table), on(d,c), on(e,b),
  on(f,d), on(g,table).
}
```

Initial mental state of agent
Inspecting the Belief Base

- \( \text{bel}(\varphi) \) succeeds if \( \varphi \) follows from the belief base in combination with the knowledge base.

Example:

\[\text{bel(clear(a), not(on(a,c)))} \text{ succeeds}\]
Combining Beliefs and Goals

- **Achievement goals:**
  \[ \text{a-goal}(\varphi) = \text{goal}(\varphi), \neg(\text{bel}(\varphi)) \]

- Useful to express that a block \(X\) is misplaced:
  \[ \text{goal}(\text{tower}([X|T])), \neg(\text{bel}(\text{tower}([X|T]))) \]

- A misplaced block is an ***achievement goal***:
  \[ \text{a-goal}(\text{tower}([X|T])) \]
Actions Change the Environment...

\[
\text{move}(a, d)
\]
Selecting Actions: Action Rules

• Action rules are used to define a strategy for action selection.

• Defining a strategy for blocks world:
  – If constructive move can be made, make it.
  – If block is misplaced, move it to table.

• What happens:
  – Check condition, e.g. can \( a\text{-}goal(tower([X|T])) \) be derived given current mental state of agent?
  – Yes, then (potentially) select \( move(X, table) \).

```plaintext
program{
  if bel(tower([Y|T])), a\text{-}goal(tower([X,Y|T])) then move(X,Y).
  if a\text{-}goal(tower([X|T])) then move(X,table).
}
```
Underspecified Programs

- Action rules may allow multiple choices of action
- Agent programs underspecify
- GOAL agent picks option randomly
- Useful for, e.g., optimizing using machine learning
An Agent is a Set of Modules

Built-in modules:

- **init** module:
  - Define global knowledge
  - Define initial beliefs & goals
  - Process “send once” percepts
  - Specify environment actions

- **main** module
  - Action selection strategy

- **event** module
  - Process percepts
  - Process messages
  - Goal management

User-defined modules.
Example Agent Program

```prolog
init module{
    knowledge{
        clear(X) :- not(on(_,X)). clear(table).
        tower([X]) :- on(X,table). tower([X,Y|T]) :- on(X,Y), tower([Y|T]).
    }
    beliefs{
        on(a,b). on(b,c). on(c,table). on(d,e). on(e,table). on(f,g). on(g,table).
    }
    goals{
        on(a,e), on(b,table), on(c,table), on(d,c), on(e,b), on(f,d), on(g,table).
    }
    actionspec{
        move(X, Y) { pre { clear(X), clear(Y), on(X,Z) } post { not(on(X,Z)), on(X,Y) } }
    }
}

main module{
    program{
        if bel(tower([Y|T])), a-goal(tower([X,Y|T])) then move(X,Y).
        if a-goal(tower([X|T])) then move(X, table).
    }
}

event module{
    ...
}
```
An Agent is a Set of Modules

Built-in modules:

- **init** module:
  - Define global knowledge
  - Define initial beliefs & goals
  - Process “send once” percepts
  - Specify environment actions

- **main** module
  - Action selection strategy

- **event** module
  - Process percepts
  - Process messages
  - Goal management

User-defined modules.
A Tooling Perspective on Agents

Developing and running an agent requires a set of different components

- Editor/Parser
- Reasoner (KRT)
- Interpreter
- Debugger
- Middleware
- Environment
- Verifier (e.g., MC)

https://github.com/eishub
https://github.com/goalhub
Building on top of existing Knowledge Representation Technologies

Second Step: Building on Top of KRTs
No Commitment to KRT

- As a design principle, the GOAL language does not commit to any KRT in particular.

Initially, built on top of Prolog.

Now:
- also OWL available, and
- a generic interface to enable flexible switching between KRTs has been developed.
Layered Language

Agent Language

program{
    if bel( clear(Y) ), a-goal( clear(Z) ) then delete(fact) + adopt(fact).
    ....
}

database (beliefs)

clear(X) :- not(on(_,X)).
clear(table).

database (goals)

query

update

KR Language, e.g., Prolog, OWL

Layered Language
Abstract definition of KRT:

A KR Technology is a 4-tuple:

\[ \langle L, \vdash, \Phi, \Theta \rangle \text{ where:} \]

- $L$ is a knowledge representation language,
- $\vdash$ is an inference relation,
- $\Phi$ is an expansion and $\Theta$ a contraction operator.
Mental State

- A mental state of agent is a triple $\langle K, \Sigma, \Gamma \rangle$ where
  - $K \subseteq L$ is the knowledge base of the agent,
  - $\Sigma \subseteq L$ is the belief base of the agent, and
  - $\Gamma \subseteq L$ is the goal base of the agent.

Mental state satisfies **rationality constraints**:

Consistency of knowledge and beliefs:
- $K \cup \Sigma$ must be consistent, i.e. it is not the case that $K \cup \Sigma \models \bot$.

Consistency of individual goals with knowledge:
- Individual goals $\gamma \in \Gamma$ must be consistent, i.e. not $K \cup \{\gamma\} \models \bot$.

Goals are rational with respect to beliefs:
- Goals $\gamma \in \Gamma$ are not believed to be true, i.e. not $K \cup \Sigma \models \gamma$. 

Mental State Conditions

• A mental state condition is a Boolean combination of \( \text{bel}(\varphi) \) and \( \text{goal}(\varphi) \) expressions.

• Example: \( \text{bel}(\varphi) \), \( \text{not}(\text{goal}(\varphi)) \)

• The semantics of a mental state condition \( \psi \) is defined on mental states \( m=\langle K, \Sigma, \Gamma \rangle \) by:
  
  - \( m \models \text{bel}(\varphi) \) iff \( K \cup \Sigma \models \varphi \)
  
  - \( m \models \text{goal}(\varphi) \) iff there is a \( \gamma \in \Gamma: K \cup \{\gamma\} \models \varphi \)
  
  - \( m \models \psi_1 \land \psi_2 \) iff \( m \models \psi_1 \) and \( m \models \psi_2 \)
  
  - \( m \models \neg \psi \) iff not: \( m \models \psi \)
KRT Interface

Agent Language

program{
    if bel( clear(Y) ), a-goal( clear(Z) ) then delete(fact) + adopt(fact).
    ....
}

Database (beliefs)

clear(X) :- not(on(_,X)).
clear(table).

Database (goals)

query

update

KR Language, e.g., Prolog, OWL, PDDL, ...

What kind of interface do we need here?
Interface: Language Abstraction

Assumes any KR language element can be mapped onto one of the following categories:

Expression

- Term
- Query
- Update
- DBFormula

Constant

Var

Function
## Interface: Functional Support

<table>
<thead>
<tr>
<th>Basic Features</th>
<th>Extra Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Parsing</td>
<td>1. Persistent storage</td>
</tr>
<tr>
<td>2. Data types (including checking)</td>
<td>2. Integrate other knowledge sources</td>
</tr>
<tr>
<td>3. Creating a store</td>
<td>3. Parallel querying</td>
</tr>
<tr>
<td>4. Modifying a store</td>
<td>4. Modularization</td>
</tr>
<tr>
<td>5. Querying</td>
<td>5. Logical validation</td>
</tr>
<tr>
<td>6. Parameter instantiation</td>
<td></td>
</tr>
<tr>
<td>7. Error handling</td>
<td></td>
</tr>
</tbody>
</table>
Embedded KR Languages
Agent Program using Prolog

init module{
knowledge{
    clear(X) :- not(on(_,X)). clear(table).
    tower([X]) :- on(X,table). tower([X,Y|T]) :- on(X,Y), tower([Y|T]).
}
beliefs{
    on(a,b). on(b,c). on(c,table). on(d,e). on(e,table). on(f,g). on(g,table).
}
goals{
    on(a,e), on(b,table), on(c,table), on(d,c), on(e,b), on(f,d), on(g,table).
}
actionspec{
    move(X, Y) { pre { clear(X), clear(Y), on(X,Z) } post { not(on(X,Z)), on(X,Y) } }
}
}
main module{
    program{
        if belief(tower([Y|T])), a-goal(tower([X,Y|T])) then move(X,Y).
        if a-goal(tower([X|T])) then move(X, table).
    }
}
event module{
    ...
}
From Prolog to PDDL (1/4)

### Prolog

```prolog
knowledge{
  clear(X) :- not(on(_,X)).
  clear(table).
  tower([X]) :- on(X,table).
  tower([X,Y|T]) :- on(X,Y), tower([Y|T]).
}
beliefs{
  on(a,b). on(b,c). on(c,table). on(d,e). on(e,table).
}
goals{
  on(a,e), on(b,table), on(c,table), on(d,c), on(e,b), on(f,d), on(g,table).
}
```

### PDDL

```pddl
knowledge{
  (impl (not (on ?z ?x)) (clear ?x)) (clear table).
  (impl (on ?x table) (tower([?x]))) ...
}
beliefs{
  (on a b) (on b c) (on c table) (on d e) (on e table) (on f g) (on g table)
}
goals{
  (and (on a e) (on b table) (on c table) (on d c) (on e b) (on f d)
       (on g table))
}
```
From Prolog to PDDL (2/4)

Prolog

init module{
  ...
  actionspec{
    move(X, Y){
      pre {clear(X), clear(Y), on(X, Z)
      post {not(on(X, Z)), on(X, Y)
    }
  }
}

PDDL

init module{
  ...
  actionspec{
    move(?x ?y){
      pre { (and (clear ?x) (clear ?y) (on ?x ?z))
      post { (and (not (on ?x ?x)) (on ?x ?y))
    }
  }
}
main module{
    program{
        if bel(tower([Y|T])), a-goal(tower([X,Y|T])) then move(X,Y).
        if a-goal(tower([X|T])) then move(X, table).
    }
}

main module{
    program{
        if bel(tower([?y|?t]) ), a-goal( (tower [?x,?y|?t]) ) then move(?x ?y).
        if a-goal( (tower [?x|?t]) ) then move(?x table).
    }
}
Adapt grammar as much to style of KR as possible?
Future Work

Extend with other KRTs:

• SQL (Datalog)
• PDDL (Planning)
• ASP (Answer Set Programming)
• Bayesian Networks (probabilistic)
• Fuzzy Logic
• …
Third Step: AI Programming
The third challenge is to continuously extend the capabilities of a programming language for decision making to allow for the development of ever more sophisticated systems, i.e., how to integrate sophisticated AI techniques.
Increasing Demand for AI

- McKinsey: by 2025, machines will be able to learn, adjust, exercise judgment, and reprogram themselves.

- Made possible by sophisticated AI techniques for: reasoning, planning, learning, and decision making.
The Next Generation AI Engineers

Artificial Intelligence

... will need to develop complex intelligent and autonomous decision-making systems

... apply complex AI techniques:
- automated reasoning
- machine learning
- automated planning
- ...

Koen Hindriks  In Control of Autonomous Decision Systems
The Next Generation AI Engineers

Al is going to make life easier for us...

... only if we make life easier for AI engineers.

Koen Hindriks  In Control of Autonomous Decision Systems
A New AI Programming Language
My aim is to design a new high-level AI programming language for autonomous decision systems that provides AI algorithms as basic building blocks.
Programming with Cognitive Modules

Cognitive Module

Artificial Intelligence
- learning
- planning

Rule-based core language
- decision rules

State components
- beliefs
- goals
- rewards
... 

State not fixed, but components defined as needed

events → action

Koen Hindriks  In Control of Autonomous Decision Systems
## Cognitive Modules and Planning

<table>
<thead>
<tr>
<th><strong>GOAL</strong></th>
<th><strong>Planning</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Knowledge</td>
<td>Axioms</td>
</tr>
<tr>
<td>Beliefs</td>
<td>(Initial) state</td>
</tr>
<tr>
<td>Goals</td>
<td>Goal description</td>
</tr>
<tr>
<td>Program Section</td>
<td>x</td>
</tr>
<tr>
<td>Action Specification</td>
<td>Plan operators</td>
</tr>
</tbody>
</table>
An Example: Integrating Reinforcement Learning (RL)

Requires expert knowledge of RL theory:

1. Create state and action representation
2. Design action selection mechanism
3. Design reward function
4. Choose update mechanism, e.g., Q-learning, prioritised sweeping, ...
5. Convergence analysis (analyse simulation runs)
6. Parameter tuning (learning rate, explore/exploit, discounts, function approximation, state representation)
Education is the next step

Teach the agent-oriented mind-set

Why? We need to train people to know how to apply our technology to ensure adoption.

Facilitate use of agent-oriented paradigm:
• Created and make available assignments and teaching materials
• Make tutorial materials widely available.
Multi-Agent Systems Project

Course Multi-Agent Systems:
Learn to program a **multi-agent system**

Develop logic-based agents programs:
- Apply reasoning technology (**Prolog**)
- Write agent programs (**GOAL**)
- Hands-on experience by various programming assignments.

Project Multi-Agent Systems:
CTF Competition in UT2004

- Control a team of bots by means of a **multi-agent system**.
- Compete at the end of the project.

Create fun assignments and projects! (**UT3, competition**)
Summary

- AOP: Programming with Mental States
- KRT: Enable flexible choice of KRT
- AI: Extend by integrating AI techniques
- Towards programming with cognitive modules
- Teaching the cognitive programming stance.