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

The Shaping of the Agent-Oriented Mindset

EMAS14 audience listed the following key concepts:

- autonomy
- rational
- goal-directedness
- interaction
- social
- reactive/events
- environment
- robustness
- decentralization
- Intentional stance

Engineering Approaches?

Theories of intelligent agents: How do the various components of an **agent's cognitive makeup** conspire to produce **rational behaviour**?

intentions
time, desires, beliefs, goals
situated automata
logical models of agents
executing agent specs
(bounded) rationality

Architectures for intelligent agents: What **structure** should an artificial intelligent agent have?

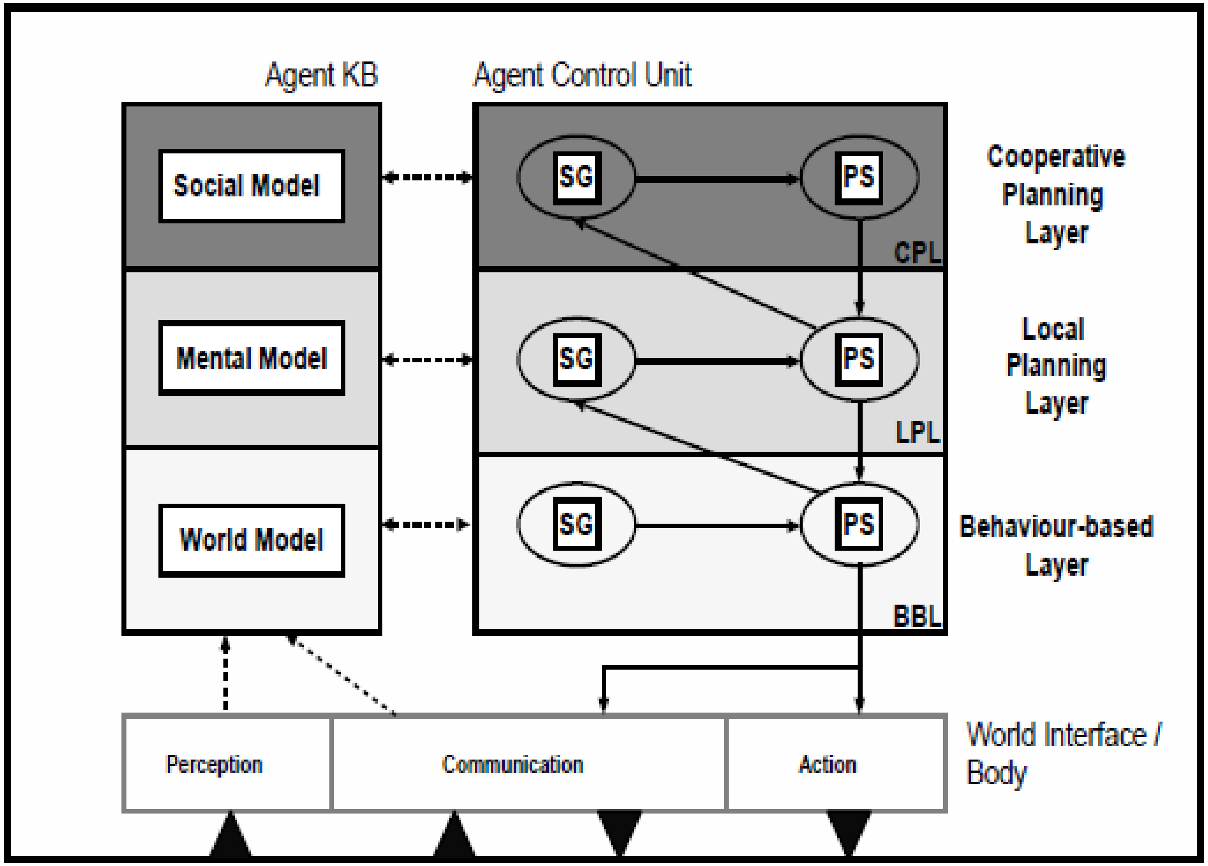
deliberative architectures
reactive architectures
hybrid architectures

Languages for intelligent agents: What are **the right primitives** for programming an intelligent agent?

agent spec languages
agent-oriented paradigm
non-logical agent languages
agent-based computing

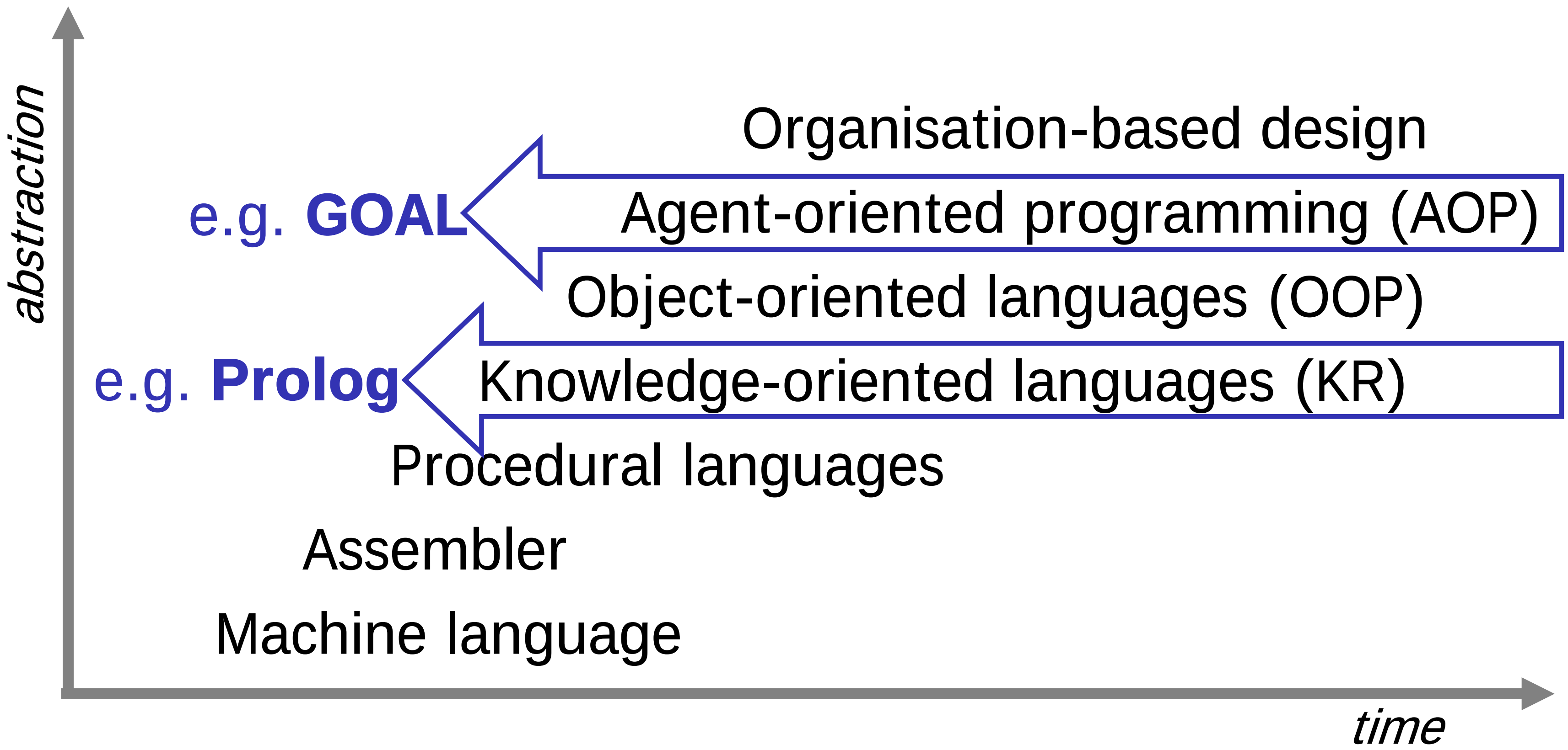
Illustrative Architecture: InteRRaP

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

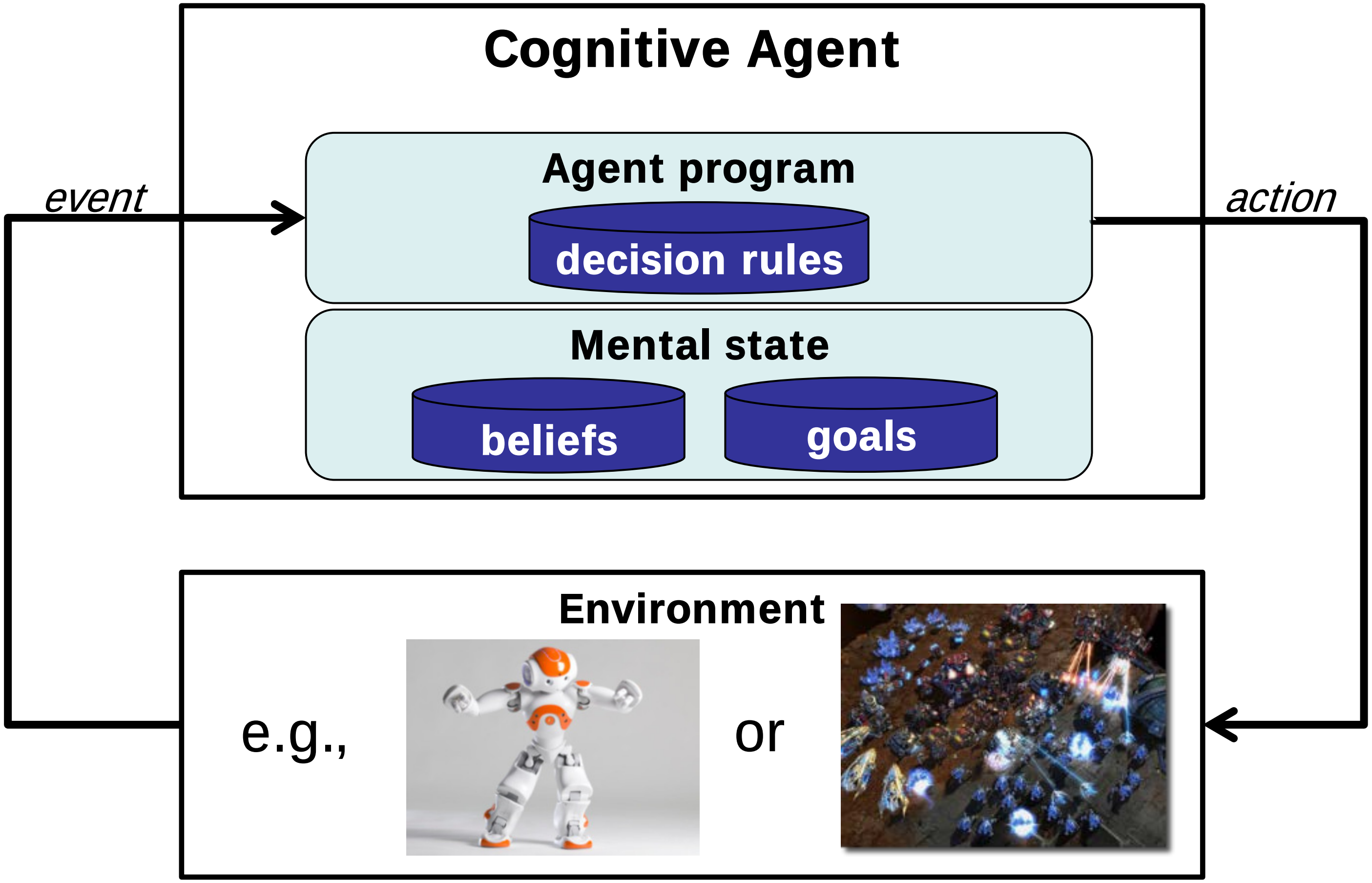


- **dMARS** architecture (MAS extension of PRS)
- Early work on coordination & organizations

Evolution of Programming

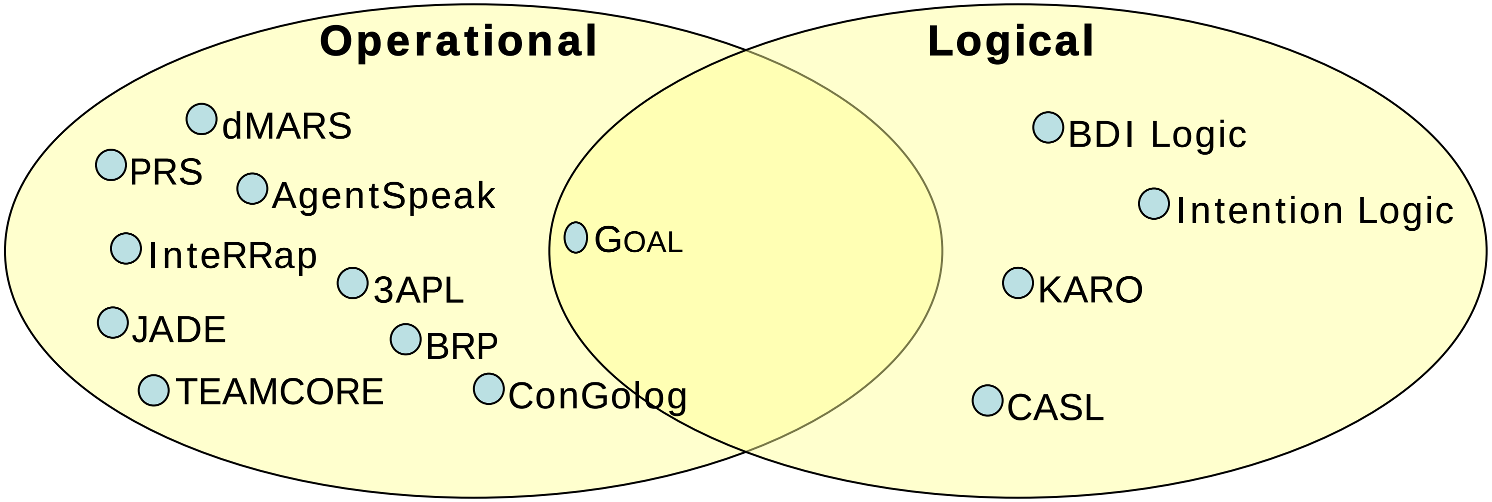


Programming with Mental States



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)

Families of Languages

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

AgentSpeak(L), Jason[2]

AGENT-0[1] (PLACA)

Golog = 3APL[3]

Main addition: Declarative goals

2APL ≈ 3APL + GOAL

Logic Programming

METATEM

Java-based Cognitive Agent Languages

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

Mobile Agents

CLAIM

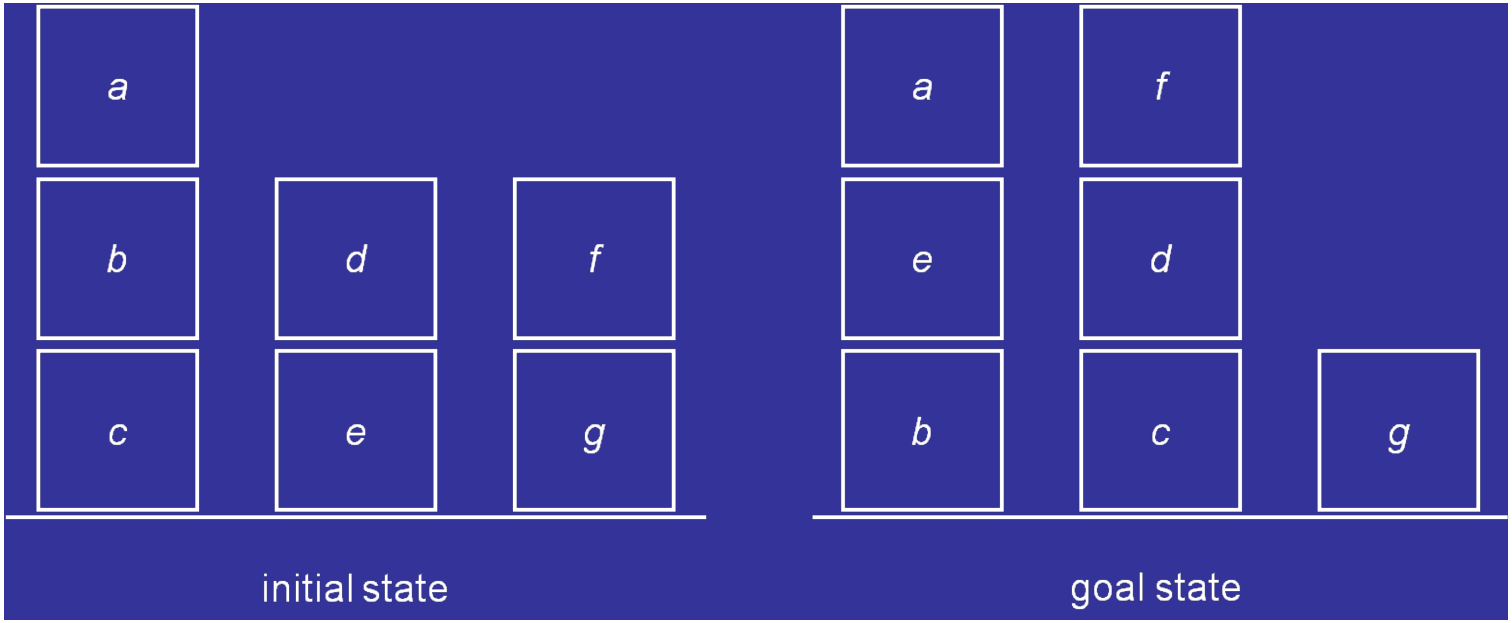
[1] mainly interesting from a historical point of view
[2] from a conceptual point of view, we identify AgentSpeak(L) and Jason
[3] without practical reasoning rules

*Introducing **GOAL***

Blocks World Toy Example

The Blocks World

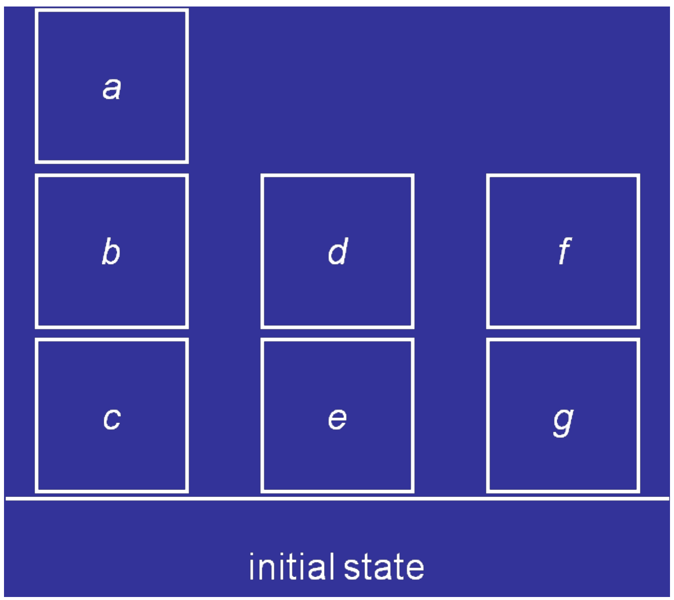
Objective: Move blocks in initial state such that result is goal state.



- *Positioning* of blocks on table is not relevant.
- A block can be moved *only if* it 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.***



Prolog

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

Initial belief base of agent

Domain Knowledge

- Domain knowledge is added to the knowledge base.

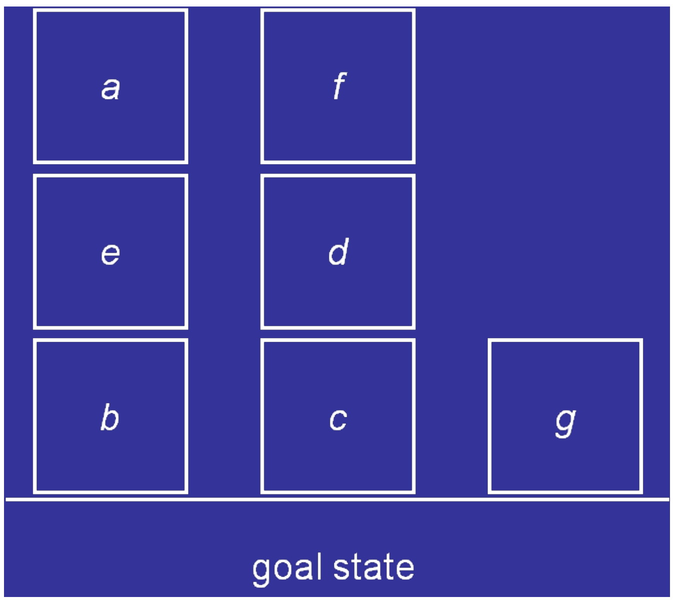
```
tower([X]) :- on(X,table).
tower([X,Y|T]) :- on(X,Y),tower([Y|T]).
clear(X) :- block(X), not(on(Y,X)).
clear(table).
```

```
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

Representing the Goal State

Using the* on(X,Y) *predicate we can represent 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

- `bel(φ)` succeeds if φ follows from the *belief base in combination with the knowledge base*.

```
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).
}
```

- Example:
  - `bel(clear(a), not(on(a,c)))` succeeds

Combining Beliefs and Goals

- *Achievement goals*:

  a-goal(φ) = goal(φ), not(bel(φ))

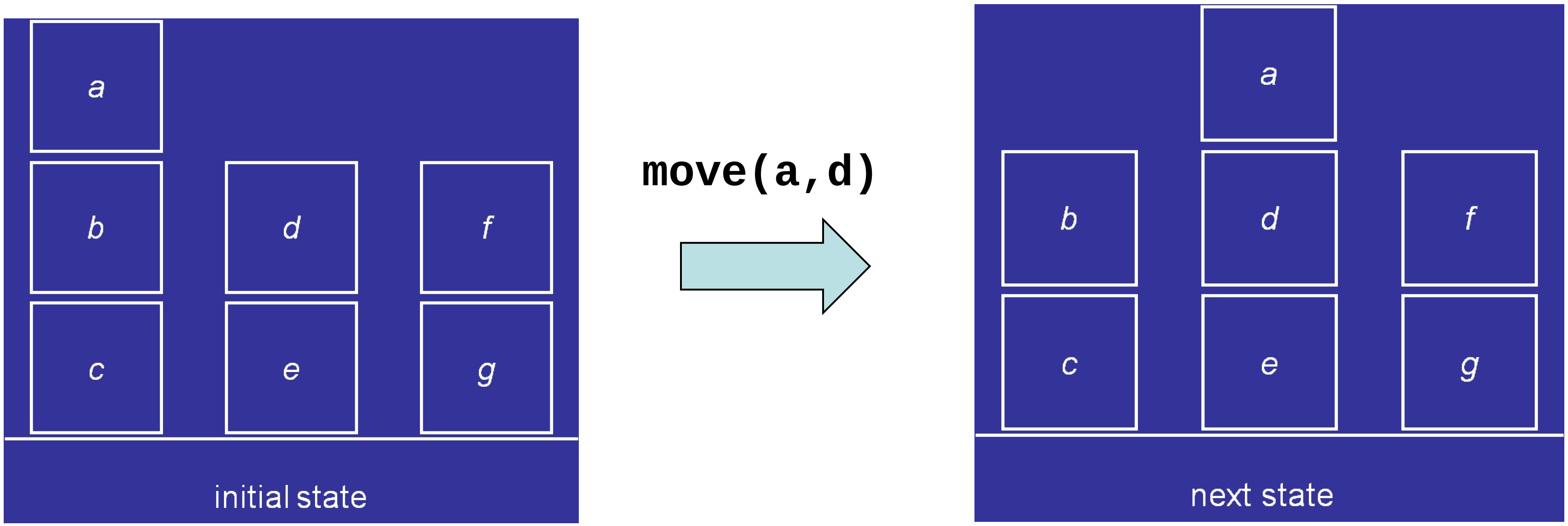
- Useful to express that a block **X** is misplaced:

  goal(tower([X|T])),not(bel(tower([X|T]))).

- A misplaced block is an *achievement goal*:

  a-goal(tower([X|T])).

Actions Change the Environment…



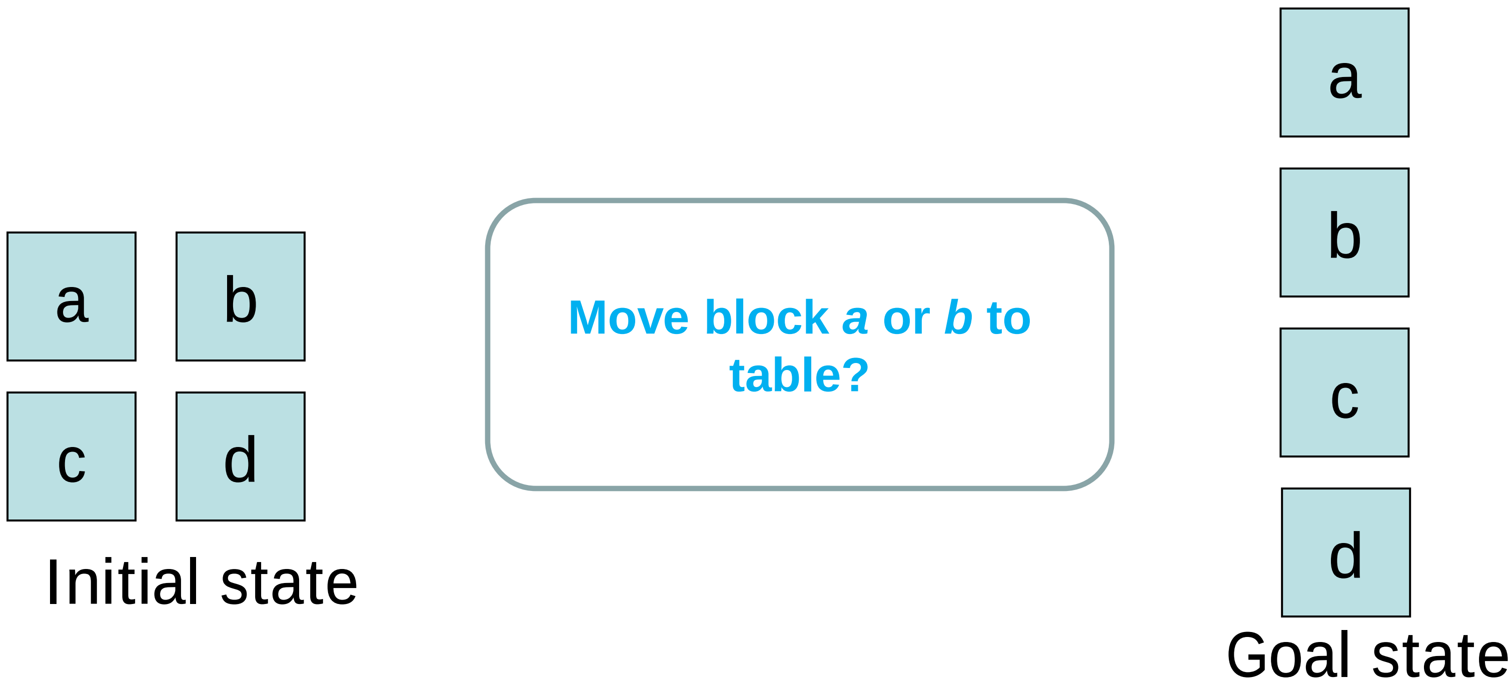
Selecting Actions: Action Rules

- 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.

```
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).
}
```

- What happens:
  - Check condition, e.g. can **`a-goal(tower([X|T]))`** be derived given current mental state of agent?
  - Yes, then (potentially) select **`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.

```
init module{
  knowledge{
    …
  }
  beliefs{
    %%% INITIAL BELIEFS ONLY IN INIT MODULE %%%
  }
  goals{
    …
  }
  program{
    %%% PROCESS “SEND ONCE” PERCEPTS HERE %%%
  }
  actionspec{
    %%% SPECIFY ENVIRONMENT ACTIONS HERE %%%
  }
}

main module{
  % OPTIONAL knowledge section
  % NO beliefs section HERE!
  % OPTIONAL goal section (not advised in ‘main’)
  program{
     %%% ENVIRONMENT ACTION SELECTION HERE %%%
  }
}

event module{
  program{
    %%% PROCESS PERCEPTS HERE %%%
    %%% PROCESS MESSAGES HERE %%%
    %%% PERFORM GOAL MANAGEMENT HERE %%%
  }
}
```

Example Agent Program

```
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.

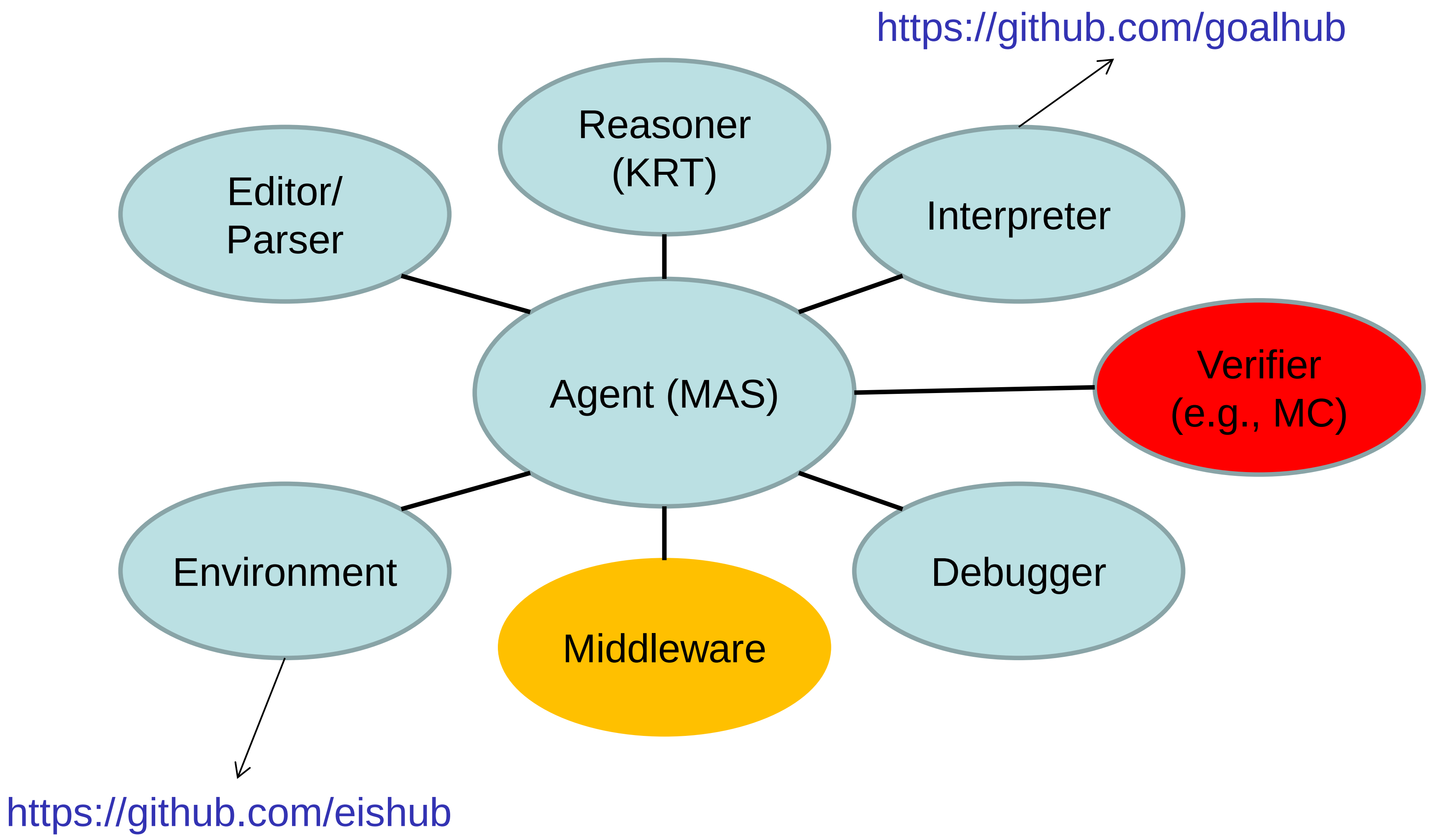
```
init module{
  knowledge{
    …
  }
  beliefs{
    %%% INITIAL BELIEFS ONLY IN INIT MODULE %%%
  }
  goals{
    …
  }
  program{
    %%% PROCESS “SEND ONCE” PERCEPTS HERE %%%
  }
  actionspec{
    %%% SPECIFY ENVIRONMENT ACTIONS HERE %%%
  }
}

main module{
  % OPTIONAL knowledge section
  % NO beliefs section HERE!
  % OPTIONAL goal section (not advised in ‘main’)
  program{
     %%% ENVIRONMENT ACTION SELECTION HERE %%%
  }
}

event module{
  program{
    %%% PROCESS PERCEPTS HERE %%%
    %%% PROCESS MESSAGES HERE %%%
    %%% PERFORM GOAL MANAGEMENT HERE %%%
  }
}
```

A Tooling Perspective on Agents

Developing and running an agent requires a set of different components

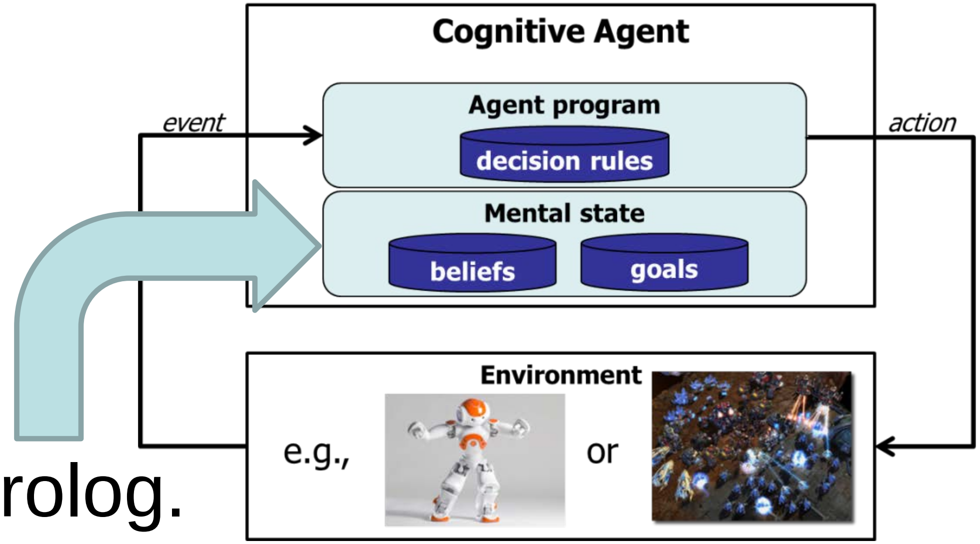


*Building on top of existing **K**nowledge **R**epresentation **T**echnologies*

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.

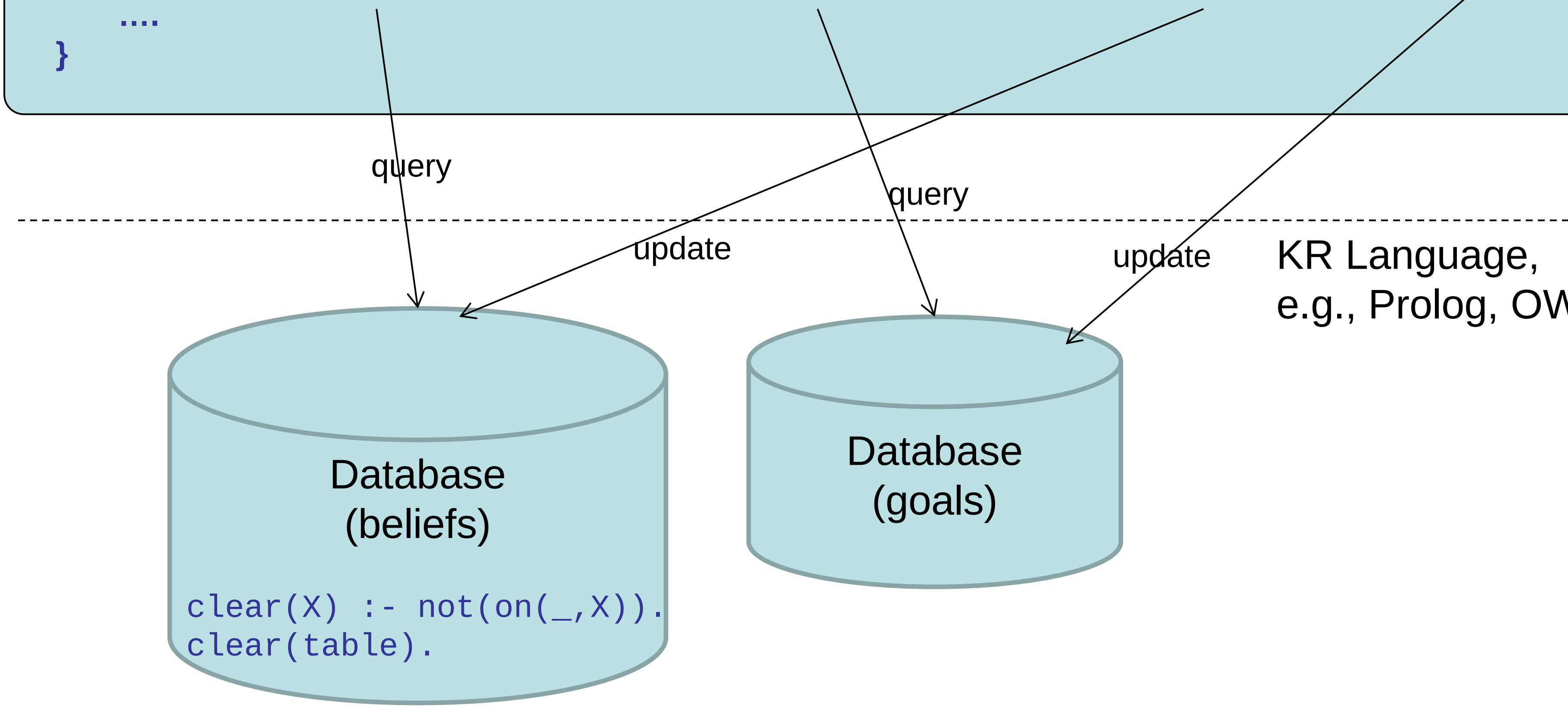


- Initially, built on top of Prolog.
- Now:
  - also OWL available, and
  - a generic interface to enable flexible switching between KRTs has been developed.

Layered Language

Agent Language

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



KR Language,
e.g., Prolog, OWL

Semantic Abstraction of KRT

Abstract definition of KRT:

A KR Technology is a 4-tuple:

$\langle \mathcal{L}, \models, \oplus, \ominus \rangle$ where:

- $\mathcal{L}$ is a knowledge representation language,
- $\models$ is an inference relation,
- $\oplus$ is an expansion and $\ominus$ 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 $K \cup \Sigma \models \perp$.

Consistency of individual goals with knowledge:

– Individual goals $\gamma \in \Gamma$ must be consistent, i.e. not $K \cup \{\gamma\} \models \perp$.

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 **`bel(`**φ**`)`** and **`goal(`**φ**`)`** expressions.

- Example: bel(φ), **`not(goal(`**φ**`))`**

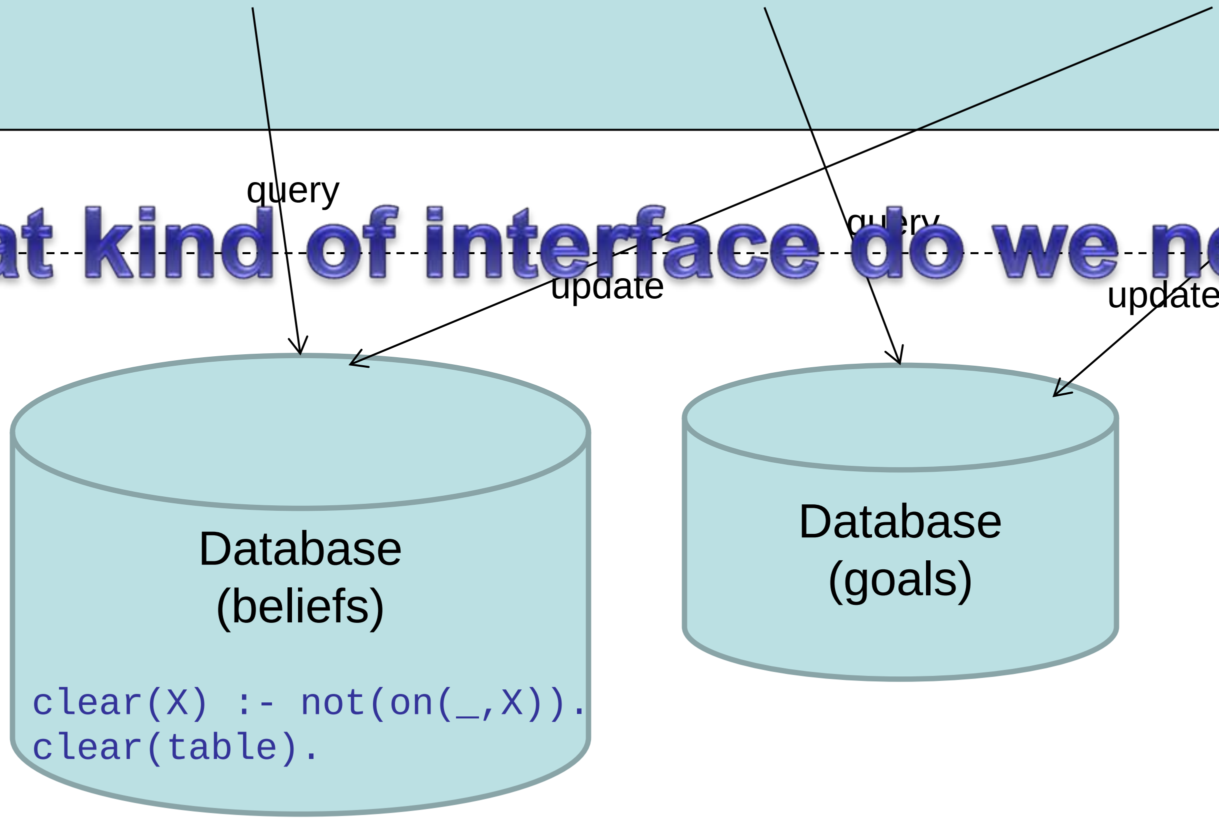
- The semantics of a mental state condition ψ is defined on mental states $m = \langle K, \Sigma, \Gamma \rangle$ by:
  - $m \models$ **`bel(`**φ**`)`** iff $K \cup \Sigma \models \varphi$
  - $m \models$ **`goal(`**φ**`)`** iff there is a $\gamma \in \Gamma$: $K \cup \{\gamma\} \models \varphi$
  - $m \models \psi_1 \wedge \psi_2$ 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).
    ....
}
```

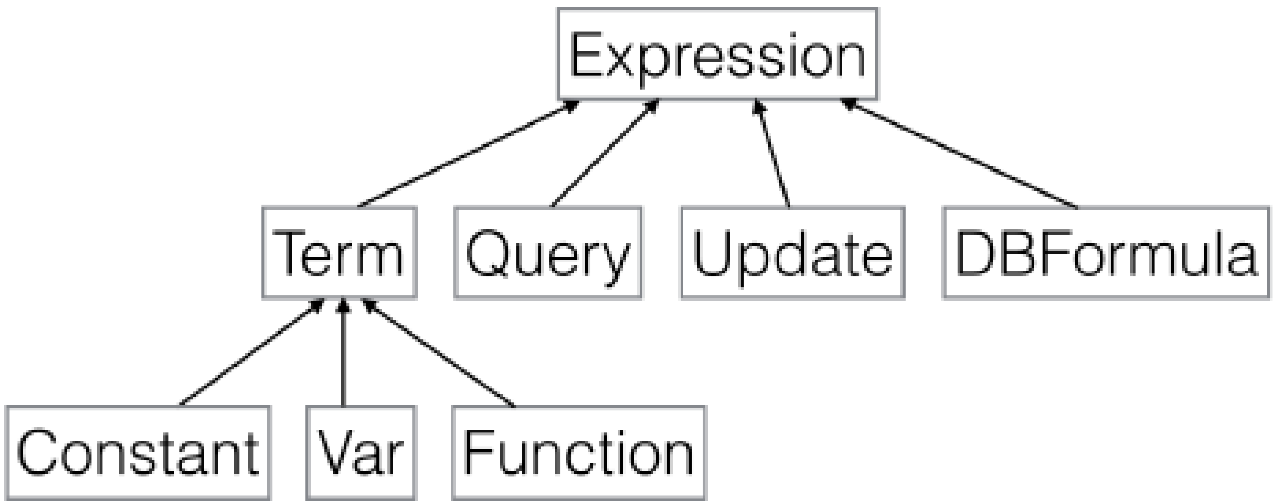
What kind of interface do we need here?



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

Interface: Language Abstraction

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



Interface: Functional Support

Basic Features

1. Parsing
2. Data types (including checking)
3. Creating a store
4. Modifying a store
5. Querying
6. Parameter instantiation
7. Error handling

Extra Features

1. Persistent storage
2. Integrate other knowledge sources
3. Parallel querying
4. Modularization
5. Logical validation

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 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{
  ...
}
```

From Prolog to PDDL (1/4)

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). 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).
```
}

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)) }
    }
  }
}
```

From Prolog to PDDL (3/4)

Prolog

```
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).
 }
}
```

PDDL

```
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).
 }
}
```

From Prolog to PDDL (4/4)

Prolog

```
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).
 }
}
```

PDDL

```
main module{
 program{
   if (and (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

AI Programs

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

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

... apply complex AI techniques:

- automated reasoning
- machine learning
- automated planning
- ...

The Next Generation AI Engineers

AI
rtificial ntelligence

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

+

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

PL
rogramming anguage

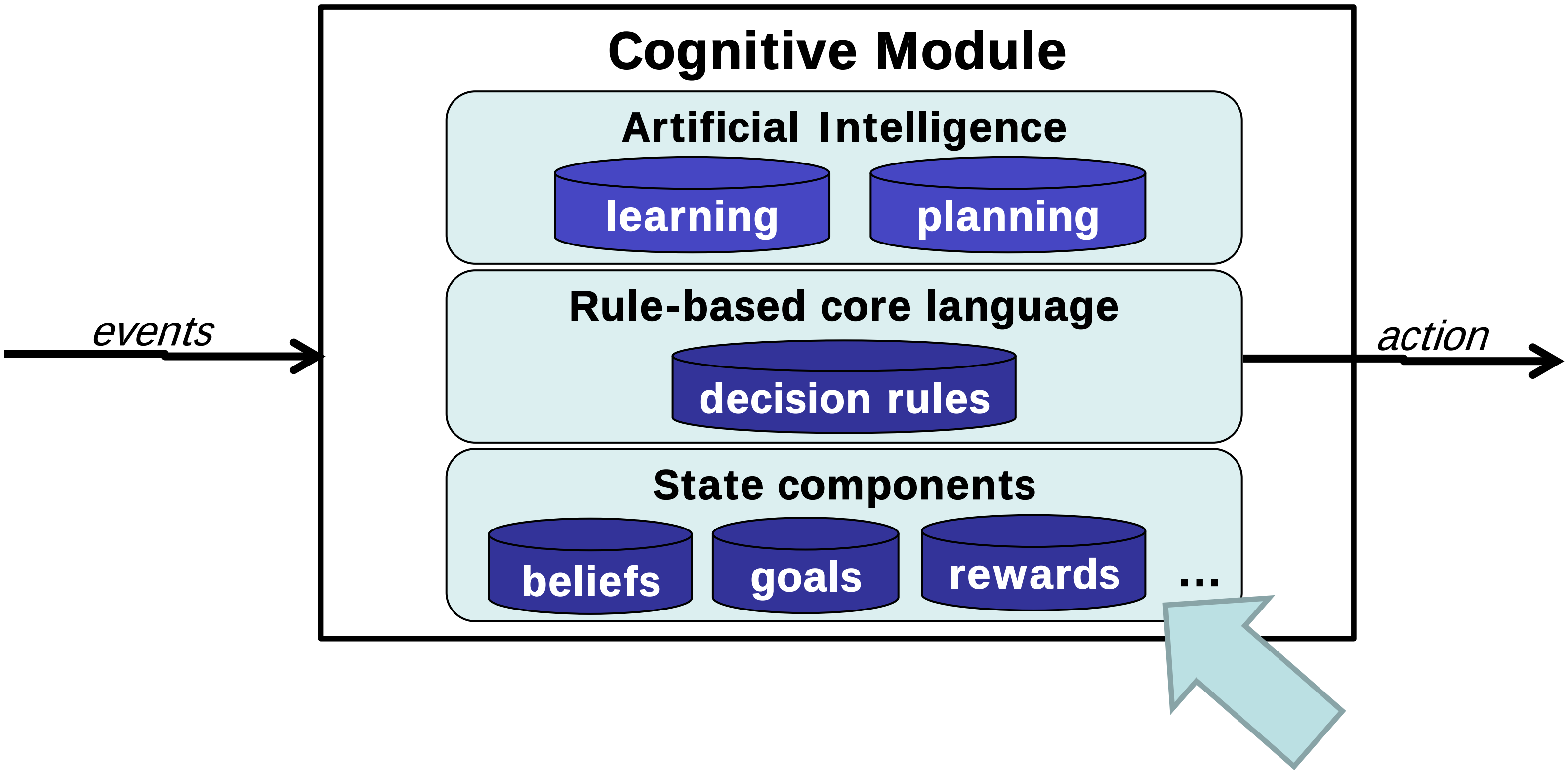
A New AI Programming Language

Cognitive Modules as Building Blocks

Cognitive agent technology offers a powerful solution for developing the next generation autonomous decision-making systems

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



State not fixed, but components defined as needed

Cognitive Modules and Planning

GOAL	Planning
• Knowledge	• Axioms
• Beliefs	• (Initial) state
• Goals	• Goal description
• Program Section	• x
• Action Specification	• Plan operators

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)

stackBuilder.goal

```
% Decide on an action to perform in Blocks World.
main module [exit=nogoals] {
    program [order=adaptive] {
        #define misplaced(X) a-goal(tower([X| T])).
        #define constructiveMove(X,Y) a-goal(tower([X, Y| T])), bel(tower([Y| T])).

        if constructiveMove(X, Y) then move(X, Y).
        if misplaced(X) then move(X, table).
    }
}
```

Education is the next step

Teach the agent-oriented mind-set

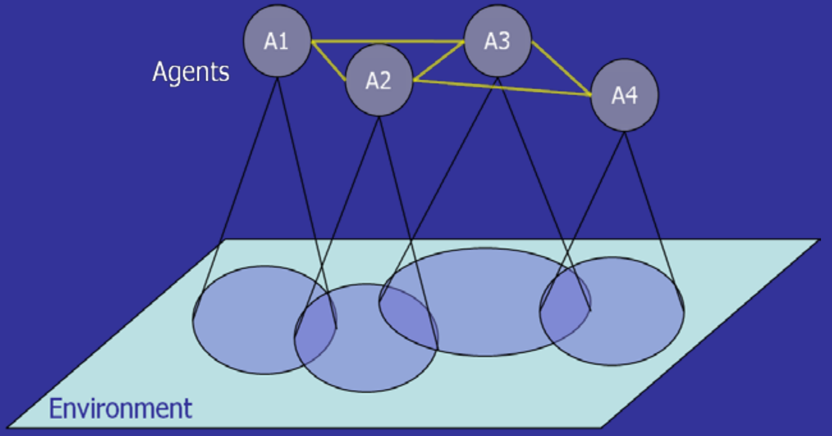
Why? We need to train people to know how to apply our technology to ensure adoption.

Facilitate use of agent-oriented paradigm:

- Created and make available assignments and teaching materials
- Make tutorial materials widely available.

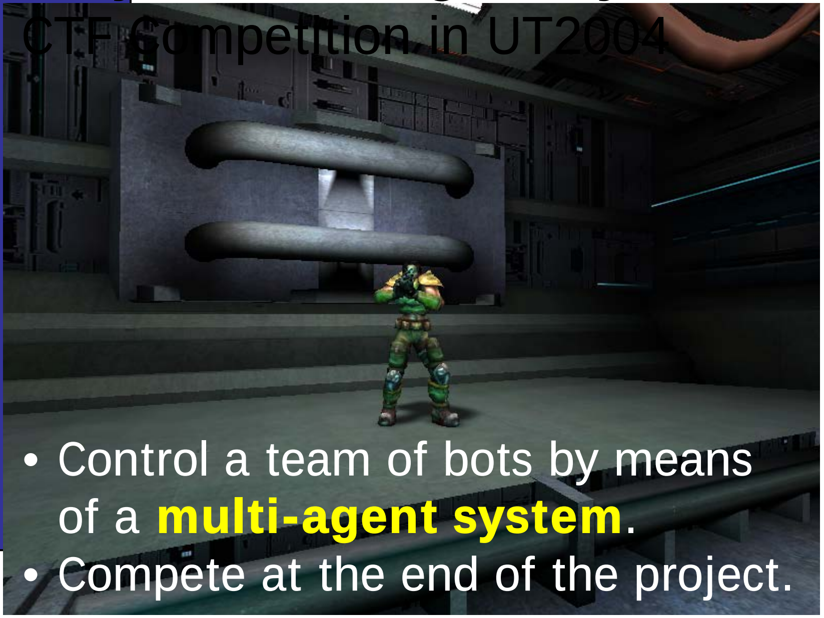
Multi-Agent Systems Project

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.