# The Agent Programming Language 3APL

Koen Hindriks
Utrecht University
email: koenh@cs.uu.nl
homepage: www.cs.uu.nl/people/koenh

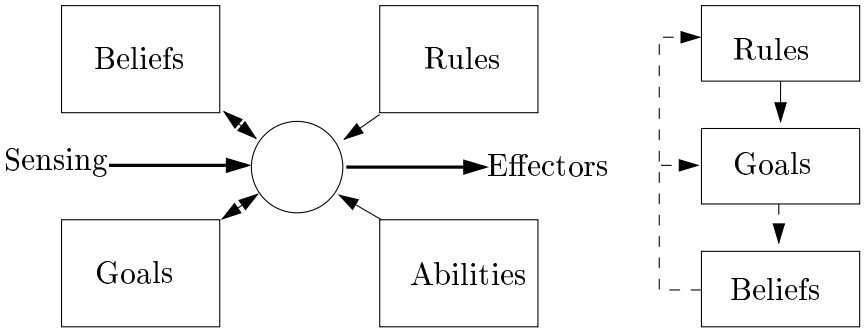
# Rule-Based Agent Programming Languages

## The Agent Languages

- AGENT-0 (Shoham'93),
- AgentSpeak(L) (Rao'96), and
- 3APL (Hindriks et al.'97,'99)

belong to the class of Rule-Based Agent Languages.

The Basic Architecture of Agents in these languages:



# The Language AgentSpeak(L)

AgentSpeak(L) is an abstraction of the system PRS, which is short for Procedural Reasoning System.

Basic Concepts of AgentSpeak(L):

- beliefs,
- simple goals - procedure calls,
- events - signaling the need for a plan,
- plan rules - $\langle goal \leftarrow plan \rangle$,
- intentions - stack of adopted plans.

AgentSpeak(L) can be embedded into 3APL:

$$\begin{aligned} \text{beliefs} &\Rightarrow \text{beliefs}, \\ \text{goals, intentions, events} &\Rightarrow \text{'complex' goals}, \\ \text{plan rules} &\Rightarrow \text{practical reasoning rules}. \end{aligned}$$

3APL is simpler+more features.

# AGENT0 vs. 3APL

| AGENT0 | 3APL |
|---|---|
| beliefs | beliefs |
| commitments | basic actions/goals |
| - | complex goals |
| commitment rules | practical reasoning rules |
| communication | communication |
| time | - |

Complex Goals in 3APL:

- basic actions,
- tests,
- sequential composition,
- while loops,
- nondeterministic choice,
- parallel composition (concurrency),
- practical reasoning rules.

# Introduction:
# A Definition of Intelligent Agents

**A Symbolic Intelligent Agent is defined by:**

- a complex mental state incorporating:
  - beliefs, representing the environment of the agent,
  - goals, representing the plans and states of affairs to be achieved,
- a set of mechanisms to manipulate this state:
  - to execute goals,
    (control over the environment),
  - for decision-making or practical reasoning,
    (means-end reasoning and goal revision).
- a set of capabilities, i.e. basic actions, which define the goals the agent can achieve.

# The Rule-Based Agent Programming Language 3APL

In the agent language 3APL these facilities are provided for by:

- an agent program, and
- a control architecture.

An agent program consists of:
- a set of capabilities,
- an initial belief base,
- a set of initial goals, and
- a set of practical reasoning rules.

# Decision-Making and Control Structure

Control Structure:
Instance of the Sense-Plan-Act Cycle.

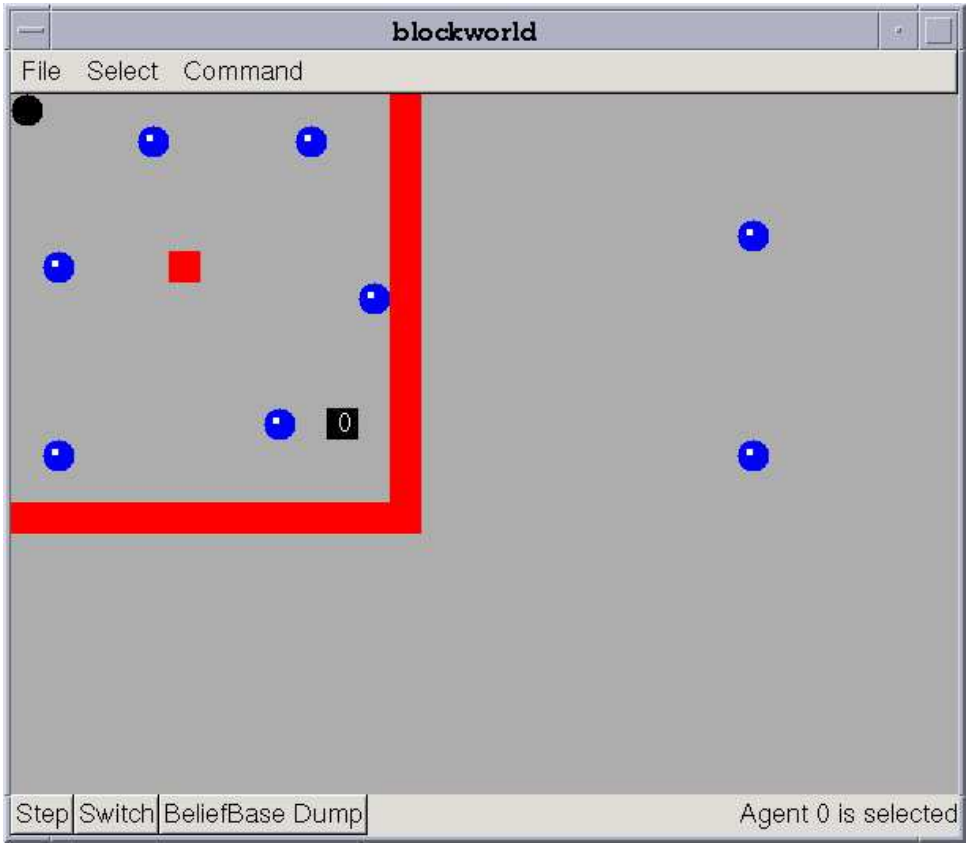
This means:

> Planning or Decision-Making
> is separated from
> Execution or Acting

Rule-Based Architectures:
Rules are used for Decision-Making.

> Decision-Making = Rule Application

# The Block World

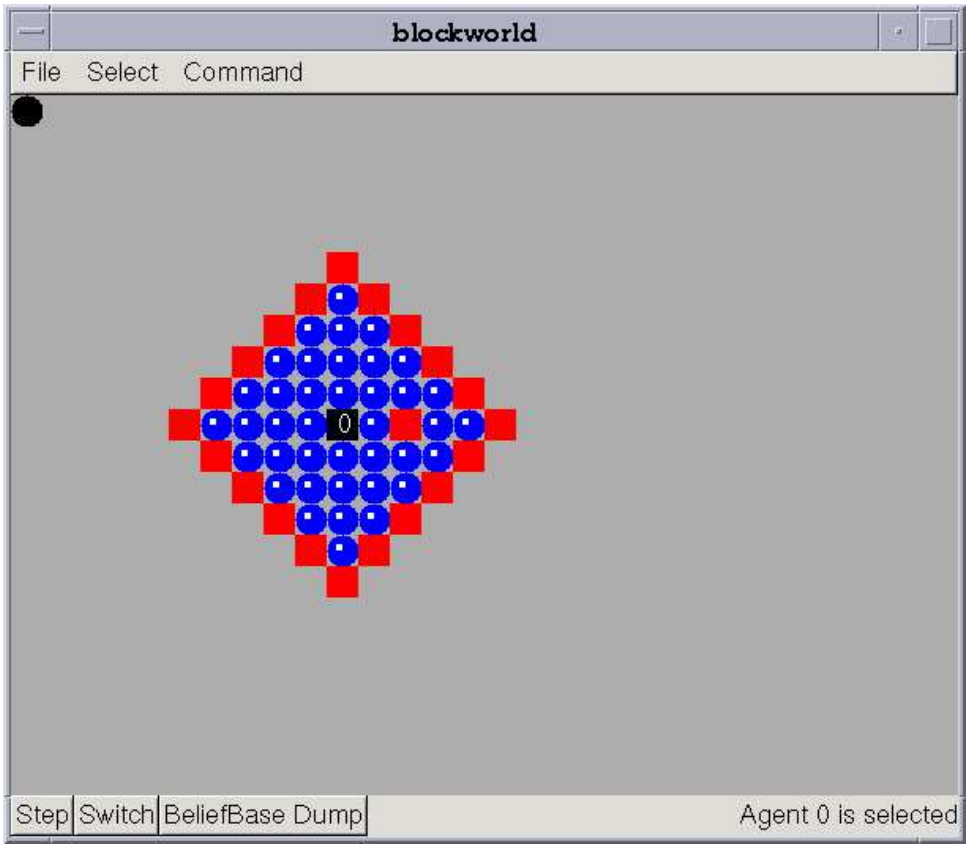



Objects in the Block World:
- The Home Base: position $0, 0$,
- Bombs in blue,
- Stones in red,
- Robot 0.

User can remove and add bombs and stones during execution.

# Sensor Range in the Block World

The robot's view is limited.




- Sensor Range = 4,
- The robot can look over walls/stones.

Belief Base:

bomb(r0,10,6). bomb(r0,9,7). etc.
stone(r0,12,10). robot(r0,10,10).

# The Explore World Task

Specification of the Explore World Task:

> The robot should explore its environment and clean up any bombs available. Cleaning up bombs means bringing bombs to the home base where they are disabled. The robot may assume the 'maze complexity' is low.

**Remark**: The robot should try to collect bombs efficiently.

# Basic Predicates in the Block World

- predicate $stone(X, Y)$,
- predicate $bomb(X, Y)$,
- predicate $robot(R, X, Y)$.

Each Cycle of the Interpreter the predicates $stone$ and $bomb$ are updated:

- visual range is taken into account,
- stones and bombs that are no longer visible are forgotten.

Other predicates can be introduced:
Example: $carriesBomb(R)$, $visited(R, X, Y)$.

Names for objects:
- a name for the robot: $r0$,
- directions: $north, east, south, west$.

# Prolog and The Closed World Assumption

Prolog is used to implement beliefs of the robot.

Initial Beliefs: A Prolog Database:

```
BELIEFBASE: robot(r0, 5, 5)
```

CWA of Prolog is inherited:
If an agent does not believe a fact, then it believes the negation of that fact.
Example:

$stone(3,3)$ is not a belief
$\Rightarrow$ NOT$stone(3,3)$ is a belief.

Using Prolog:
we can write Prolog programs to compute all kinds of things and define new predicates.

# Example Prolog Definitions

## A Prolog Clause for Computing 'nearestBomb':

```
nearestBomb(R, X, Y) :-
    bombs(R, L1, L2), robot(R, X0, Y0),
    minimum(L1, L2, X0, Y0, X, Y).
```

## A Clausal Definition of 'blocked(dir)':

```
blocked(R,north) :- robot(R,X,Y), is(Z,Y-1),
                    stone(R,X,Z).
blocked(R,north) :- robot(R,X,Y), Y-1<0.
blocked(R,east):- robot(R,X,Y), is(Z,X+1),
                    stone(R,Z,Y).
blocked(R,east) :- robot(R,X,Y), X+1>29.
blocked(R,south) :- robot(R,X,Y), is(Z,Y+1),
                    stone(R,X,Z).
blocked(R,south) :- robot(R,X,Y), Y+1>22.
blocked(R,west) :- robot(R,X,Y), is(Z,X-1),
                    stone(R,Z,Y).
blocked(R,west) :- robot(R,X,Y), X-1<0.
```

# Specifying Agent Capabilities in 3APL

A specification of an action `A` should specify:
- when it is possible to execute the action, and
- what the effects of executing the action are.

Components of 3APL Action Specifications:

- name of primitive action,
- preconditions of action
  = delete list,
- postconditions of action
  = add list.

Actions are updates on the belief base:
- The delete list is removed from the belief base,
- The add list is added to the belief base.

# Primitive Actions Supported by the Block World: Movement Actions

A specification of the action `North(R)`:

```
{ robot(R,X,Y), is(Z,Y-1), NOT stone(R,X,Z) }
 North(R)
{ robot(R,X,Y-1), visited(R,X,Y-1) }
```

Failure of Actions:

- Precondition Failure: The action is not executed,
- Blockworld Failure: The agent believes the action can be executed, but in fact it cannot.

# Primitive Actions Supported by the Block World: Pickup(R) and Drop(R)

A specification of the action `Pickup(R)`:

```
{ bomb(R, X, Y), robot(R, X, Y), NOT
carriesBomb(R) }
 Pickup(R)
{ robot(R, X, Y), carriesBomb(R) }
```

A specification of the action `Drop(R)`:

```
{ carriesBomb(R), robot(R, X, Y) }
 Drop(R)
{ robot(R, X, Y), bomb(R, X, Y) }
```

# Actions Independent of the Block World

Actions to change the belief base can be freely defined:

A specification of the action `InsFree(R,X,Y)`:

```
{ true() }
 InsFree(R,X,Y)
{ free(R,X,Y) }
```

A specification of the action `DelFree(R,X,Y)`:

```
{ true() }
 DelFree(R,X,Y)
{ NOT free(R,X,Y) }
```

# Writing a 3APL Program

A 3APL agent consists of:

- A Set of Prolog Clauses, ✓
- An Initial Belief Base, ✓
- A Goal Base,
- A Rule Base.

A Goal Base consists of plans:

```
GOALBASE:
robotLookAround(r0);cleanupBombs(r0)
```

# Practical Reasoning Rules
# Robot Look Around

A Practical Reasoning Rule consists of

- A head: any goal or plan,
- A guard: a condition of the belief base,
- A body: any goal or plan.

Practical Reasoning Rules for looking Around:

```
robotLookAround(R) <- robot(R, X, Y) |
 lookAround(R, X, Y, north);
 lookAround(R, X, Y, east);
 lookAround(R, X, Y, south);
 lookAround(R, X, Y, west),

lookAround(R, X, Y, south) <-
   is(Z, Y + 1) AND [Z < 29]
   AND [NOT visited(R,X,Z)] |
   InsFree(R, X, Z),
```

# Rules for Cleaning up Bombs

Two Cases:
- The robot knows location of bomb,
- The robot does not know location of bomb.

```
cleanupBombs(R) <- nearestBomb(R, X, Y) |
    getBomb(R, X, Y);
    bomb(R, X, Y)?; /* dynamic world! */
    Pickup(R);
    exploreWorld(R, 0, 0);
    Drop(R);
    cleanupBombs(R),

cleanupBombs(R) <- nearestFree(R, X, Y) |
    exploreWorld(R, X, Y);
    cleanupBombs(R)
```

# Rules for Exploring World

Priorities on Rules:

```
exploreWorld(R, X, Y) <-
   bomb(R, X0, Y0) AND NOT carriesBomb(R) | SKIP,

exploreWorld(R, X, Y) <- robot(R, X, Y) | SKIP,

exploreWorld(R, X, Y) <- robot(R, X0, Y0) |
   IF Y < Y0 AND [NOT visited(R, X0, Y0 - 1)]
      THEN north(R)
   ELSE IF X0 < X AND [NOT visited(R, X0 + 1, Y0)]
      THEN east(R)
   ELSE IF Y0 < Y AND [NOT visited(R, X0, Y0 + 1)]
      THEN south(R)
   ELSE IF X < X0 AND [NOT visited(R, X0 - 1, Y0)]
      THEN west(R);
   exploreWorld(R, X, Y),
```

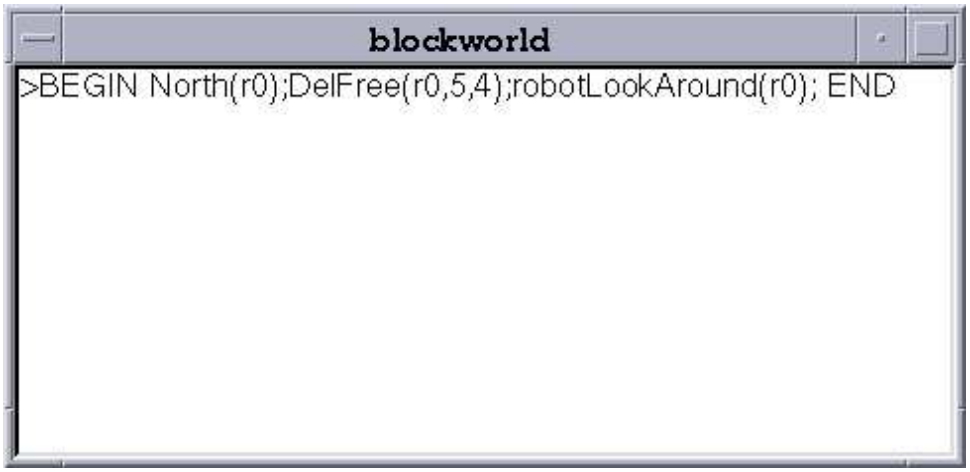
# Movement Actions Again

Retracting predicate 'free':

```
north(R) <- robot(R, X, Y) |
   North(R); DelFree(R, X, Y - 1); robotLookAround(R),
```

Failing basic actions:

```
North(R); DelFree(R, X, Y) <-
   blocked(R, north) |
   east(R),
```

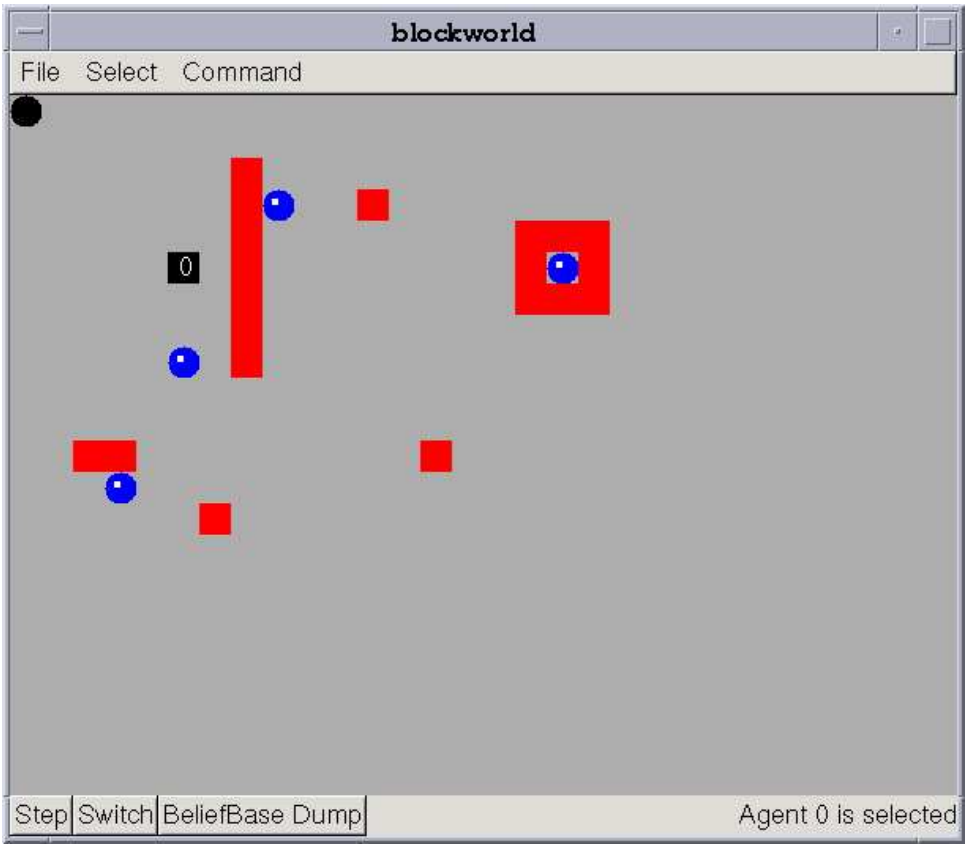


# Monitoring a Goal

Suppose the robot has a goal

```
getBomb(r0,8,3)
```

but there is a bomb closer than the target bomb.



# Implementing an Interrupt

Labelling a Goal:

- introduce a label,
- postfix this to the goal to be monitored.

```
getBomb(r0,X,Y); l(r0, X, Y)
```

Implement Interrupt with Rule with Goal Variable:

```
G; l(R,X0,Y0) <- nearestBomb(R,X,Y) AND [X<X0]|
getBomb(R,X,Y);l(R,X,Y),
```

# Rules in 3APL
# A Classification

Types of rules in 3APL:

- Failure Rules:
  `North(R)<- blocked(R,north) | East.`
- Reactive Rules:
  `<- blocked(R) | evasiveMove(R),`
- Means-End Rules:
  `exploreWorld` rule,
- Optimisation Rules:

```
G; l(R,X0,Y0) <- nearestBomb(R,X,Y) AND [X<X0]|
getBomb(R,X,Y);l(R,X,Y),
```

# Conclusions

- 3APL is a powerfull agent programming language,
- It supports the construction of 'complex' goals and plans,
- Complex Practical Reasoning rules,
- No commitment to the KR-language,
- We did not discuss communication,
- For more information:

http://www.cs.uu.nl/people/koenh