Incremental Reasoning Agent

Till Hofmann, Matthias Loebach, Tim Niemueller

1 Problem Statement

2 Knowledge Base & Inference Engine

3 CLIPS

4 World Model

5 Reaction and Interaction with the World

6 Summary

7 Hands-On

Overview

1 Problem Statement

2 Knowledge Base & Inference Engine

3 CLIPS

4 World Model

5 Reaction and Interaction with the World

6 Summary

7 Hands-On

Problem Statement

The robot has a strategic mission (maximal score)

The mission can only be achieved with the right decisions

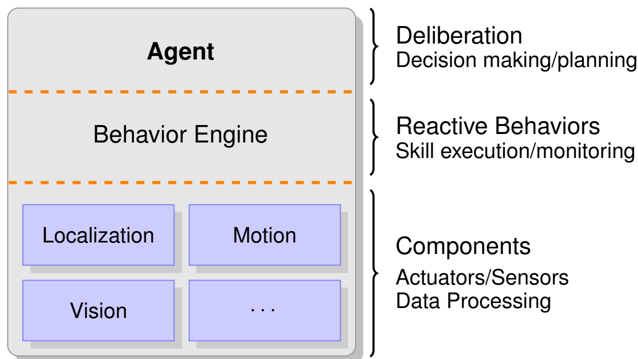
- High complexity and dynamic of production chains
- Incomplete knowledge
- Plan breaks due to non-deterministic events
- Communication breakdowns
- Robot breakdowns

Why an Expert System?

World State Representation Through Knowledge Base

- Readability:
  - Readable representation of the current world state
  - Situation specific rules
  - Rules are non-scoped (no if-else constructs)
- Transparency: reasoning is easily understandable
- Adaptability: simple extension of the world model

Behavioral Architecture

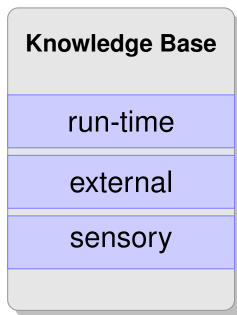


Overview

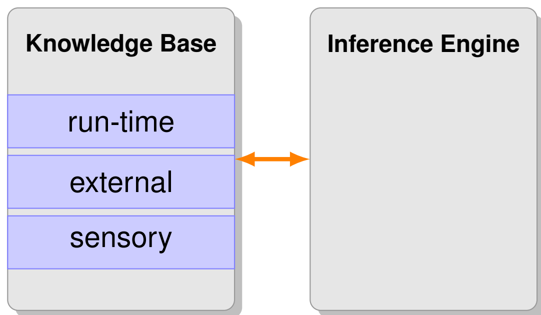
1 Problem Statement
2 Knowledge Base & Inference Engine
3 CLIPS
4 World Model
5 Reaction and Interaction with the World
6 Summary
7 Hands-On

Knowledge Base & Inference Engine

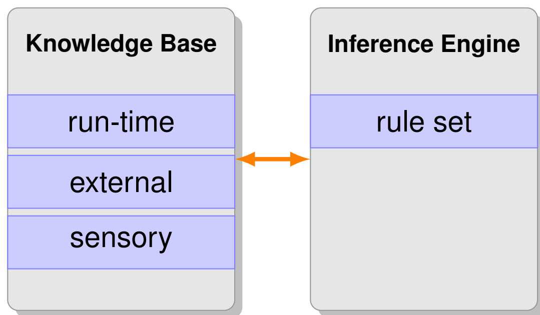
Knowledge Base & Inference Engine



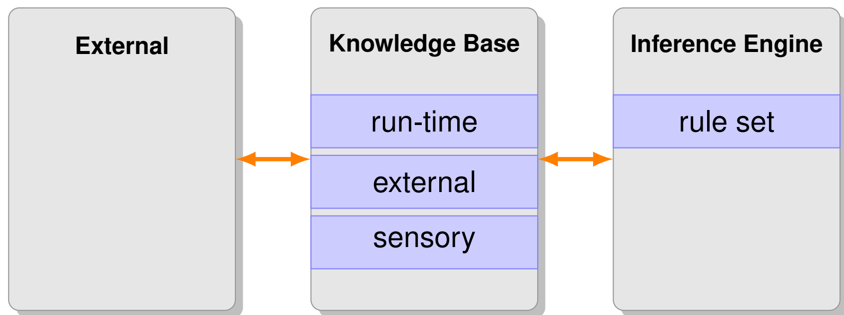
Knowledge Base & Inference Engine



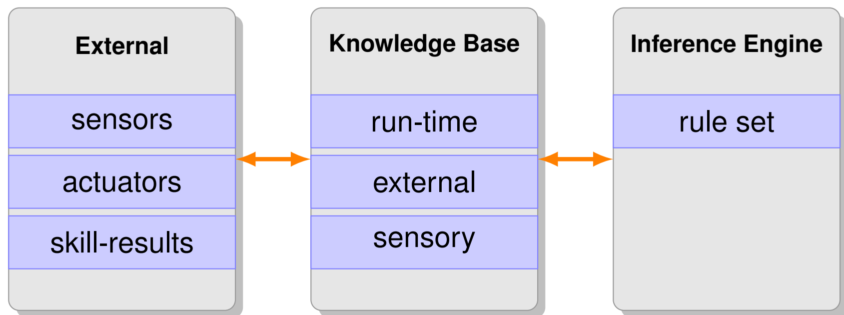
Knowledge Base & Inference Engine



Knowledge Base & Inference Engine



Knowledge Base & Inference Engine



Scope of the Knowledge Base

(co-operative) agent information

- Machine location and type
- Locked positions

Scope of the Knowledge Base

(co-operative) agent information

- Machine location and type
- Locked positions

Game information

- Orders
- Machine state
- Game time

Scope of the Knowledge Base

(co-operative) agent information

- Machine location and type
- Locked positions

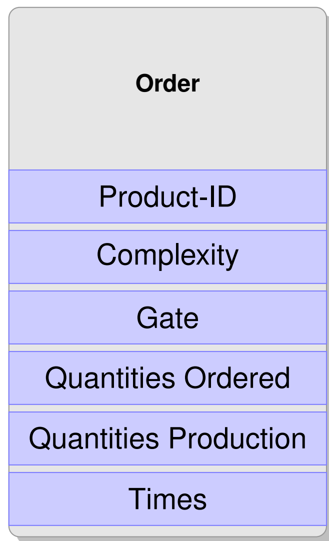
Game information

- Orders
- Machine state
- Game time

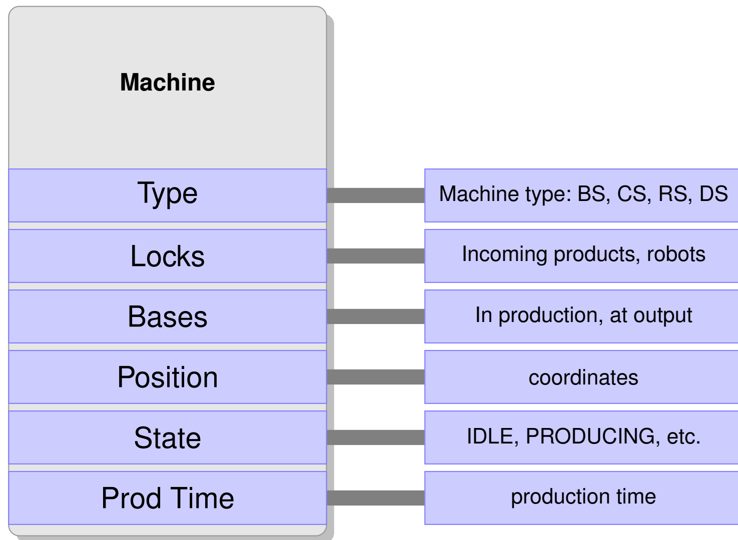
Sensory information

- Tag detection
- Gripper state

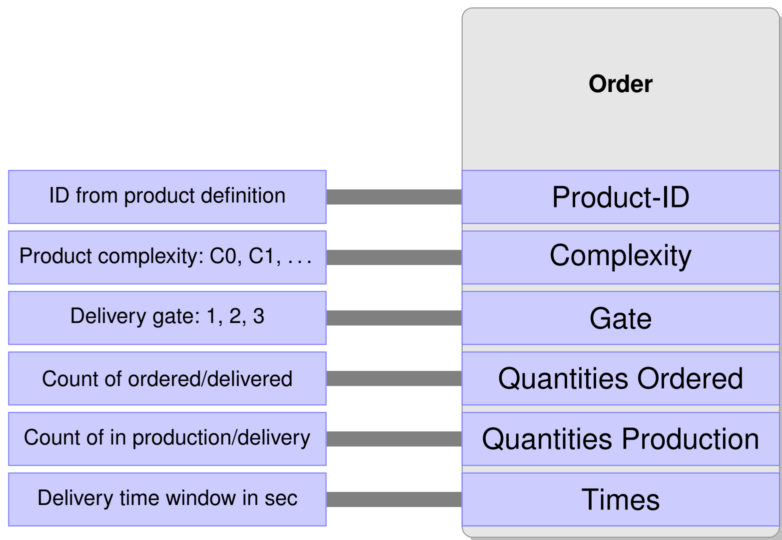
Representing World State in the Knowledge Base



Representing World State in the Knowledge Base



Representing World State in the Knowledge Base



Overview

1 Problem Statement
2 Knowledge Base & Inference Engine
3 CLIPS
4 World Model
5 Reaction and Interaction with the World
6 Summary
7 Hands-On

CLIPS – A Rule-based Production System

Rule-based Production Systems

- First-Order Logic forward chaining systems
- Productions: condition-action rules
- Working memory holds facts (“short-term memory”)
- Rules encode heuristic knowledge (“long-term memory”)

C Language Integrated Production System – CLIPS

- Graph-based Rete-Algorithm
- Lisp-style syntax
- Typically large rule bodies and relatively small number of facts
  - Number of rules: 224
  - Number of fact templates: 49
  - Number of fact changes: >200000 in 10 minutes
- Integrates nicely with C/C++

CLIPS Terminology

Facts Information in Working Memory

```
(machine (name M-CS1) (mtype CS) ...)
```

- Incomplete Knowledge
  - Explicitly, e.g. `(mtype UNKNOWN)`
  - Non-existence of facts

CLIPS Terminology

Facts Information in Working Memory

```
(machine (name M-CS1) (mtype CS) ...)
```

- Incomplete Knowledge
  - Explicitly, e.g. `(mtype UNKNOWN)`
  - Non-existence of facts

Functions Procedural Knowledge

```
(deffunction get-output (?mps)
  "Return the navgraph point of the output side of
      the given mps"
  (return (str-cat ?mps "-O"))
)
```

CLIPS Terminology

Rules Heuristic Knowledge

```
(defrule rule-name
  ?m <- (machine (name C-DS))
  =>
   (modify ?m (mtype DS))
)
```

CLIPS Terminology

Rules Heuristic Knowledge

```
(defrule rule-name
  ?m <- (machine (name C-DS))
  =>
  (modify ?m (mtype DS))
)
```

Agenda Currently active rules

- List of all rules that have their conditions met
- Rules with higher salience are placed higher on the agenda
- Only one rule active (executed) at a time

Overview

1 Problem Statement

2 Knowledge Base & Inference Engine

3 CLIPS

4 World Model

5 Reaction and Interaction with the World

6 Summary

7 Hands-On

World Model - Overview

Ordered Facts

```
(game-time 217 771447.0)
(team-color CYAN)
(holding NONE)
(phase PRODUCTION)
(state IDLE)
```

Structured Facts

```
(ring (color BLUE) (req-bases 1))
```

World Model - Machines

Simple Machine Fact

```
(machine (name C-BS) (team CYAN) (mtype BS)
  (incoming) (incoming-agent) (loaded-id 0)
  (produced-id 0) (state IDLE))
```

World Model - Machines

Simple Machine Fact

```
(machine (name C-BS) (team CYAN) (mtype BS)
  (incoming) (incoming-agent) (loaded-id 0)
  (produced-id 0) (state IDLE))
```

Cap Station Fact

```
(machine (name M-CS2) (team MAGENTA) (mtype CS)
  (incoming) (incoming-agent) (loaded-id 0)
  (produced-id 611270821) (state READY-AT-OUTPUT))

(cap-station (name M-CS2) (cap-loaded NONE)
  (assigned-cap-color BLACK))
```

World Model - Orders

Order Fact

```
(order (id 1) (product-id 821171164)
  (complexity C0) (delivery-gate 3)
  (quantity-requested 1) (quantity-delivered 0)
  (begin 0) (end 900)
  (in-production 0) (in-delivery 0))
```

World Model - Products

Product Order Fact

```
(product (id 821171164) (product-id 0)
  (rings) (cap BLACK) (base RED))
```

Real Product Fact

```
(product (id 611270821) (product-id 821171164)
  (rings) (cap NONE) (base RED))
```

Overview

1 Problem Statement

2 Knowledge Base & Inference Engine

3 CLIPS

4 World Model

5 Reaction and Interaction with the World

6 Summary

7 Hands-On

Rule Components

Formulate rules to

- Process information
- Trigger skills
- Communicate with other agents
- Check world model sanity

Rule Components

Formulate rules to

- Process information
- Trigger skills
- Communicate with other agents
- Check world model sanity

Rule Syntax

```
(defrule <rule-name> [<comment>]
  [<declaration>]           ; Rule Properties
  <conditional-element>* ; Left-Hand Side (LHS)
  =>
  <action>*                 ; Right-Hand Side (RHS)
)
```

Binding of Variables

Bind variables in rules to

- Match facts against each other
- Modify fact
- Re-use their value

Binding of Variables

Bind variables in rules to

- Match facts against each other
- Modify fact
- Re-use their value

Binding and matching variables

```
(defrule example
  (team-color ?team)
  ?m <- (machine (name C-RS2) (team ?team)
    (loaded-id ?id&~NULL))
  =>
  <action>
)
```

Changing the World – Adding New Facts

Which knowledge has to be added to the fact base?

- New task to start a production step
- Correction for broken world model
- Locks from other agents

Changing the World – Adding New Facts

Which knowledge has to be added to the fact base?

- New task to start a production step
- Correction for broken world model
- Locks from other agents

Assert New Fact

```
(assert (task (state proposed)) ...)
```

Changing the World – Cleaning Up

Why remove old facts?

- Obsolete facts clutter the knowledge base (e.g. old steps)
- Facts may represent an old world state (e.g. a base in the gripper)
- Rules might fire continuously due to unchanged conditions

Changing the World – Cleaning Up

Why remove old facts?

- Obsolete facts clutter the knowledge base (e.g. old steps)
- Facts may represent an old world state (e.g. a base in the gripper)
- Rules might fire continuously due to unchanged conditions

Retract a Fact

The fact is specified via the bound fact address.

```
(retract ?step)
```

Changing the World – Modifying Facts

Existing facts have to be constantly updated with knowledge

- Most new knowledge is only partial
- New knowledge mostly changes existing facts
- Facts can be selectively modified instead of re-writing
- Increases readability and maintainability
- Executes a retract and an assert
- No modify for *ordered facts!*

Changing the World – Modifying Facts

Existing facts have to be constantly updated with knowledge

- Most new knowledge is only partial
- New knowledge mostly changes existing facts
- Facts can be selectively modified instead of re-writing
- Increases readability and maintainability
- Executes a retract and an assert
- No modify for *ordered facts!*

Modifying a Fact

The fact is specified via the bound fact address. If the fact is structured only fields being modified have to be listed.

```
(modify ?task (state running))
```

Changing the World – Conditional changes

The right-hand side may also contain simple conditions

- If the additional conditional element is only a single value
- If additional facts have to be asserted in some cases
- Avoids clutter with a lot of similar rules

Changing the World – Conditional changes

The right-hand side may also contain simple conditions

- If the additional conditional element is only a single value
- If additional facts have to be asserted in some cases
- Avoids clutter with a lot of similar rules

Conditional Change

```
(if (not ?base) then
  (retract ?hf)
  (assert (holding NONE))
  (printout error ``Lost base during drive_to''
     crlf)
)
```

Agent Rules Explained

Full Rule with Conditions and Actions

```
(defrule discard-unneeded-base
  (declare (salience ?*DISCARD-UNKNOWN*))
  (phase PRODUCTION)
  (state IDLE)
  (team-color ?team-color&~nil)
  (holding ?product-id&~NONE)
  (product (id ?product-id))
  (machine (mtype RS) (name ?rs) (team ?team-color)
     )
  =>
  (bind ?task-id (random-id))
  (assert
    (task (name discard-unknown)
      (id ?task-id) (state proposed)
      (priority ?*DISCARD-UNKNOWN*)
      (steps (create$ (+ ?task-id 1))))
    (step (name discard) (id (+ ?task-id 1))
      (task-priority ?*DISCARD-UNKNOWN*))
  )
)
```

Overview

1 Problem Statement

2 Knowledge Base & Inference Engine

3 CLIPS

4 World Model

5 Reaction and Interaction with the World

6 Summary

7 Hands-On

Summary

Utilize the agent to make strategic decisions in the game

- Represent your knowledge in the fact base (knowledge base)
- Make decisions on the basis of the current world state
- Use the skiller to interact with the world
- Local distributed incremental agent
- Biggest opportunity for diversification in the RCLL

Overview

1 Problem Statement
2 Knowledge Base & Inference Engine
3 CLIPS
4 World Model
5 Reaction and Interaction with the World
6 Summary
7 Hands-On

Hands-On – Using CLIPS

First Task

- Recognize the arrival on the playing field
- Drive via way-points on a navigational graph
- Only drive when the game is RUNNING

Afterwards you should be able to:

- React to game phase changes
- Call skills
- Handle finished skills
- Model a production process through consecutive skill calls

Hands-On – Using CLIPS

Second Task

- Prepare a cap station with a cap from its shelf
- Remove the empty base from the cap station
- Deliver the base to a delivery station

Afterwards you should be able to:

- Pickup and insert base elements into machines
- Communicate with production machines

Hands-On: Restoring your map

You will first need to undo the changes to the Navgraph:

1. Change the world in your `~/.bashrc`:

   ```
   GAZEBO_WORLD_PATH=~/robotics/gazebo-rcll/
      worlds/carologistcs/llsf.world
   ```

2. Reload your bashrc:

   ```
   source ~/.bashrc
   ```

3. Change the config to use the correct map: In `cfg/gazsim-configurations/gazsim-config-override.yaml`, change every `map_file` to `maps/rc-2016.png`

Hands-On – Using CLIPS

Agent source code is located in

```
~/robotics/fawkes-robotino/src/agents/
```

CLIPS agent webview

http://localhost:8081/clips/agent

CLIPS Programming Guide

http://clipsrules.sourceforge.net/documentation/v630/bpg.pdf

Start the simulation with this command

```
./gazsim.bash -x start -r -n1 -t -a
```

Task 1

- Extract the archive `agent_hands-on.tar` to `~/robotics/fawkes-robotino/src/agents/`
  Now the file `~/robotics/fawkes-robotino/src/agents/summer_school2018/summer_school2018.clp` should exist
- In the file `cfg/conf.d/clips-agent.yaml`, change `agent: llsf2015/llsf2015` to `agent: summer_school2018/summer_school2018`
- Add your implemenation to `~/robotics/fawkes-robotino/src/agents/summer_school2018/task1.clp`

Hints:

- Look at the webview (CLIPS Agent) to understand the fact base!
- Have a look at the other agents, in particular `rcll2016`

Task 1 Hints, cont.

- Run the simulation and start the game, monitor the web interfaces for facts that change and that may be important
- Watch out for `change-state`, this is what you need
- To *start* a skill, call the function `skill-call`, e.g.,

```
(skill-call drive_into_field team ?team-color
    wait 0)
```

  or

```
(skill-call ppgoto place "C-BS-I")
```

- To check that a skill *finished*, check for a fact `skill-done`, e.g.,

```
?skill <- (skill-done (name "drive_into_field")
      (status FINAL|FAILED))
```

Task 2 Hints

- This is the same task as on Tuesday!
- Use the skills `bring_product_to` and `get_product_from`
- To instruct an MPS, use the fact `mps-instruction`, e.g.,:

```
(assert (mps-instruction (machine C-CS1)
  (cs-operation RETRIEVE_CAP)))
```