Overview of expert system shells

Krzysztof Kaczor, Szymon Bobek, Grzegorz J. Nalepa

Institute of Automatics
AGH University of Science and Technology, Poland

Inżynieria wiedzy
12.05.2010, Kraków

http://geist.agh.edu.pl

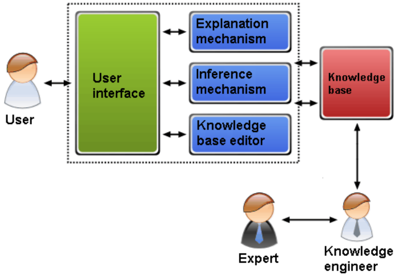
Outline

Rule–based systems

- Rules – very popular method for knowledge representation.
  - Usually presented in the **IF...THEN...** form.

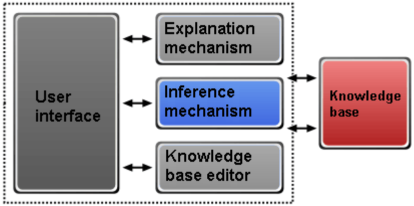
$$\underbrace{\text{IF the sun is shining}}_{\text{LHS}}\ \underbrace{\text{THEN it's warm}}_{\text{RHS}}$$

- Rule–based systems – a class of expert systems.



Expert System Shells

- Expert system shell



- Has to provide an inference engine and a rule representation language.
- Expert system shells – CLIPS, JESS, DROOLS.

CLIPS

- Supports rule-based, object-oriented and procedural programming.
- Inference engine uses the RETE algorithm.
- Only provides forward chaining.
- LISP-like syntax – all expressions are enclosed within roundz brackets.
- The rule format:

```
(defrule rule_name ’’optional_comment’’
(pattern_1) ; Left-Hand Side (LHS)
(pattern_2) ; of the rule consisting of elements
.           ; before the "=>"
.
.
(pattern_N)
=>
(action_1) ; Right-Hand Side (RHS)
(action_2) ; of the rule consisting of elements
.          ; after the "=>"
.
(action_M))
```

JESS

- Inference engine uses the RETE algorithm.
- Provides forward and backward chaining.
- Extends CLIPS syntax.
- Knowledge represented as rules or JessML:

```
(defrule
example-rule
(button)
=>

(printout t

"Hello!" crlf))
```

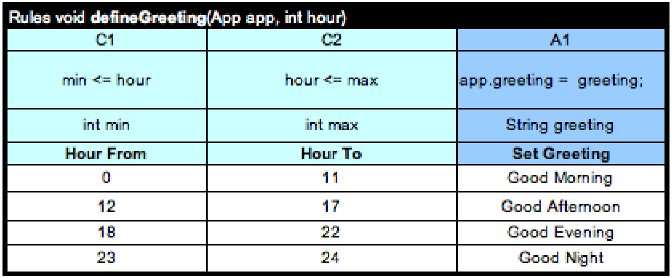
```
<rule>
  <name>myrule</name>
  <lhs>
    <group>
      <name>and</name>
      <pattern>
        <name>MAIN::button</name>
      </pattern>
    </group>
  </lhs>
  <rhs>
    <funcall>
      <name>printout</name>
      <value type="SYMBOL">t</value>
      <value type="STRING">Hello!</value>
      <value type="SYMBOL">crlf</value>
    </funcall>
  </rhs>

</rule>
```

Drools5

- More than a classic expert system shell – provides a platform for integration of processes and rules.
- Consists of four modules:
  - Drools Guvnor – knowledge base repository.
  - Drools Expert – rule engine.
  - Drools Flow – workflow modelling.
  - Drools Fusion (event processing/temporal reasoning).
- Only provides forward chaining.
- Inference engine uses a RETE-based algorithm.
- Knowledge represented as rules in Drools5 format:

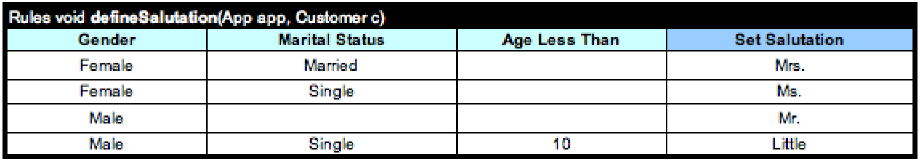
```
rule "RuleName" when
  // conditions
then
  // actions

end
```

Hello World example



Rules void defineGreeting(App app, int hour)

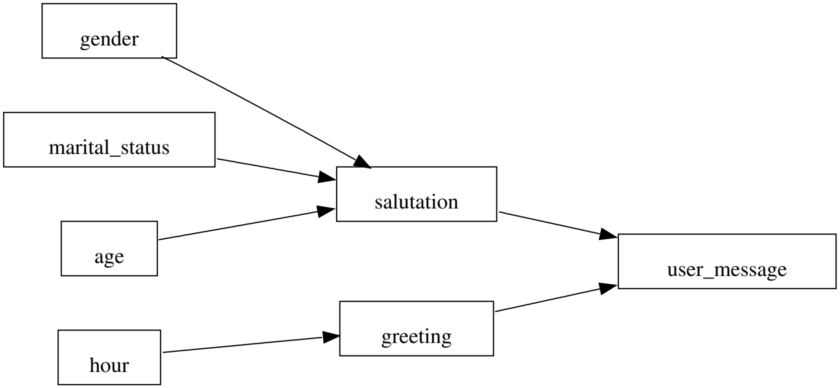
C1	C2	A1
min <= hour	hour <= max	app.greeting = greeting;
int min	int max	String greeting
Hour From	**Hour To**	**Set Greeting**
0	11	Good Morning
12	17	Good Afternoon
18	22	Good Evening
23	24	Good Night



Rules void defineSalutation(App app, Customer c)

Gender	Marital Status	Age Less Than	Set Salutation
Female	Married		Mrs.
Female	Single		Ms.
Male			Mr.
Male	Single	10	Little

Hello World example



Hello World - CLIPS

```
(defrule rule1 ""
   (gender female)
   (marital_status married)
   (age ?years)
   (test (>= ?years 10))
   =>
(assert (salut Mrs.))
(printout t "rule1 -> fired" crlf)
)

(defrule rule2 ""
   (gender female)
   (marital_status single)
   (age ?years)
   (test (>= ?years 10))
   =>
(assert (salut Ms.))
(printout t "rule2 -> fired" crlf)
)
...
```

```
...
(defrule rule8 ""
   (hour ?value)
   (test (<= ?value 24))
   (test (>= ?value 23))
=>
   (assert (greet "Good Night"))
   (printout t "rule9 -> fired" crlf)
)

(defrule rule9 ""
   (greet ?g)
   (salut ?s)
=>
   (printout t ?g " " ?s crlf)
)
```

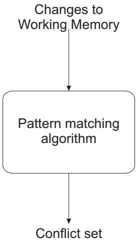
Hello World - Drools

```
rule "Morning"
  no-loop
  when
    d : DayTime(hour < 12);
  then
    d.setGreeting("Good morning");
    update( d );
end
rule "Afternoon"
  no-loop
  when
    d : DayTime(hour >= 12, hour < 18);
  then
    d.setGreeting("Good afternoon");
    update( d );
end
```

```
rule "Salutation Mr"
  no-loop
  when
    p : Person( age >= 10, male == true );
  then
    p.setSalutation("Mr.");
    update( p );
end

rule "UserMessage"
  when
    p : Person( s : salutation, salutation != null
    d : DayTime( g : greeting, greeting != null);
  then
  System.out.println(s + " " + g);
end
```

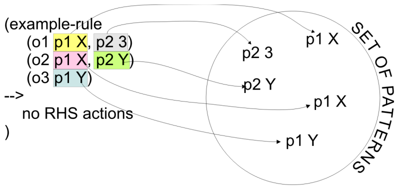
Rule Inference Algorithm

- An inference algorithm performs three steps:
  1. Pattern Matching.
  2. Conflict Set Resolution.
  3. Action Execution.
- *Pattern Matching* is a bottleneck of the inference process.
- The naive algorithm is far too slow.
- More efficient algorithms: RETE, TREAT, GATOR.

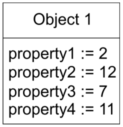


Rule Inference Algorithm – concepts

- Facts are stored in the *Working Memory*.
- LHS consists of *patterns*:



- *Network* – a tree-like structure consisting of patterns.
- *Working element* – an object with attribute/value pairs describing it.

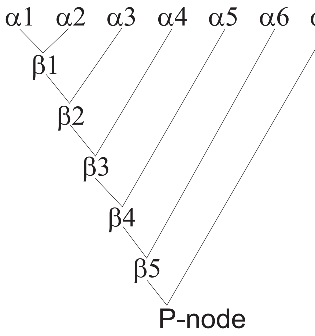


Rule Inference Algorithm – concepts

- A network contains two types of nodes:
  - *intra-nodes* – involve only one working element.
    ```
    (rule1
    (O1, p1 X, p2 12, p3 X, p4 11)
    (O2, p1 2, p2 3))
    ```
  - *inter-nodes* – involve more than one working element.
    ```
    (rule1
    (O1, p1 X, p2 12, p3 X, p4 Y)
    (O2, p1 2, p2 Y))
    ```
- *Alpha memory* – a set containing all intra-nodes (a subset of the network).
- *Beta memory* – a set containing all inter-nodes (a subset of the network).
- *Productions* – a set containing rules.
- *Conflict set* – a set containing ready-to-fire rules.

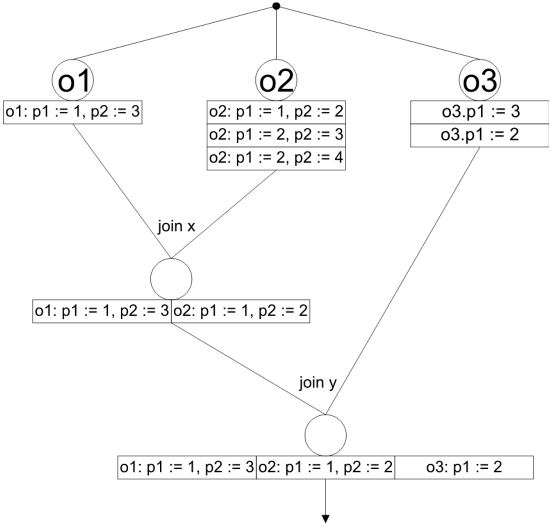
RETE

- RETE was considered the most efficient inference algorithm.
- It tries to avoid iterating over the working memory and the production set.
- Network build process: [fragile]
  1. Alpha-memory building – linear sequence of intra-elements (one-input nodes).
  2. Beta-memory building – joins of intra- and inter-elements (two-input nodes).
- Each node has a memory.



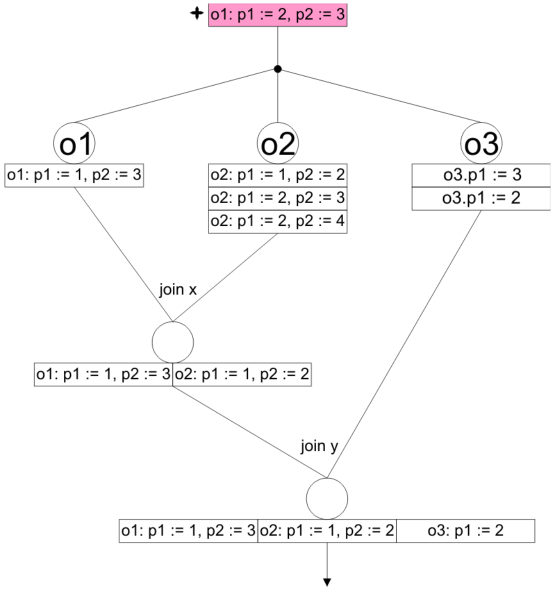
RETE

```
(rule1
   (o1: p1 X, p2 3)
   (o2: p1 X, p2 Y)
   (o3: p1 Y)
)
```



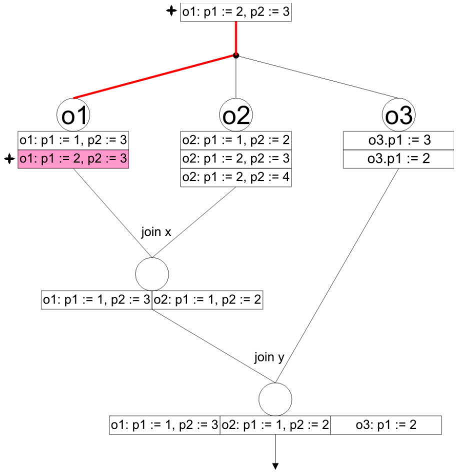
RETE

```
(rule1
   (o1: p1 X, p2 3)
   (o2: p1 X, p2 Y)
   (o3: p1 Y)
)
```



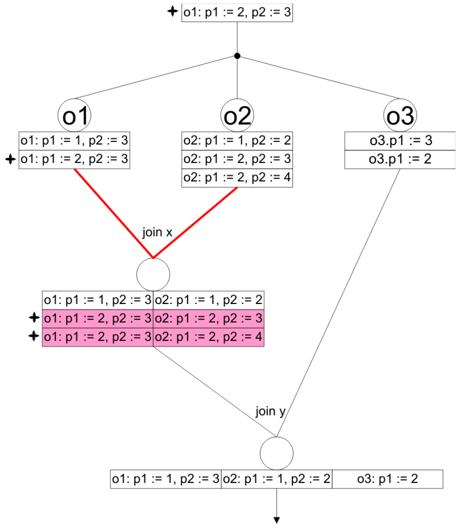
RETE

(rule1
  (o1: p1 X, p2 3)
  (o2: p1 X, p2 Y)
  (o3: p1 Y)
)



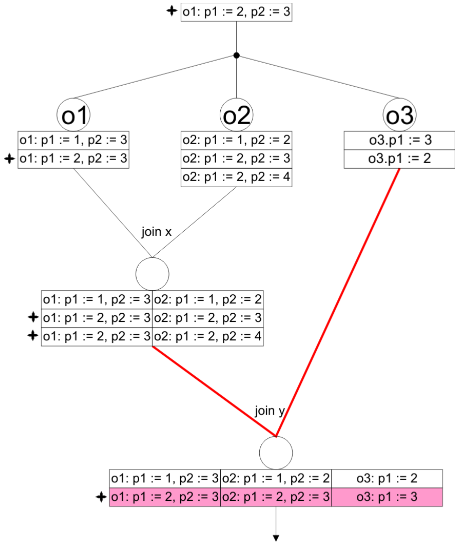
RETE

```
(rule1
  (o1: p1 X, p2 3)
  (o2: p1 X, p2 Y)
  (o3: p1 Y)
)
```


+ o1: p1 := 2, p2 := 3
o1
o2
o3
o1: p1 := 1, p2 := 3
+ o1: p1 := 2, p2 := 3
o2: p1 := 1, p2 := 2
o2: p1 := 2, p2 := 3
o2: p1 := 2, p2 := 4
o3.p1 := 3
o3.p1 := 2
join x
o1: p1 := 1, p2 := 3 | o2: p1 := 1, p2 := 2
+ o1: p1 := 2, p2 := 3 | o2: p1 := 2, p2 := 3
+ o1: p1 := 2, p2 := 3 | o2: p1 := 2, p2 := 4
join y
o1: p1 := 1, p2 := 3 | o2: p1 := 1, p2 := 2 | o3: p1 := 2

RETE

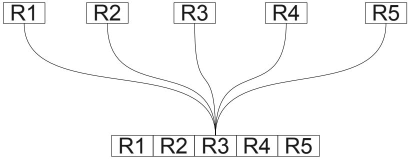
(rule1
  (o1: p1 X, p2 3)
  (o2: p1 X, p2 Y)
  (o3: p1 Y)
)



+ o1: p1 := 2, p2 := 3
o1
o2
o3
o1: p1 := 1, p2 := 3
+ o1: p1 := 2, p2 := 3
o2: p1 := 1, p2 := 2
o2: p1 := 2, p2 := 3
o2: p1 := 2, p2 := 4
o3.p1 := 3
o3.p1 := 2
join x
o1: p1 := 1, p2 := 3 | o2: p1 := 1, p2 := 2
+ o1: p1 := 2, p2 := 3 | o2: p1 := 2, p2 := 3
+ o1: p1 := 2, p2 := 3 | o2: p1 := 2, p2 := 4
join y
o1: p1 := 1, p2 := 3 | o2: p1 := 1, p2 := 2 | o3: p1 := 2
+ o1: p1 := 2, p2 := 3 | o2: p1 := 2, p2 := 3 | o3: p1 := 3

TREAT – General concepts

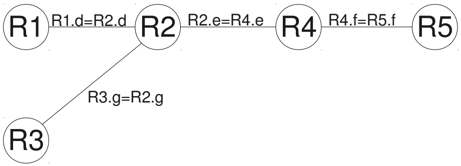
- Based on three ideas:
  1. Memory support – creation and maintenance of the alpha-memory.
  2. Conflict set support – the conflict set is explicitly retained across production system cycles.
  3. Condition membership – introduces a new property for each rule, called *rule-active*. The match algorithm ignores rules that are *non–active*.
- TREAT does not use beta-memory.



Gator – general concepts

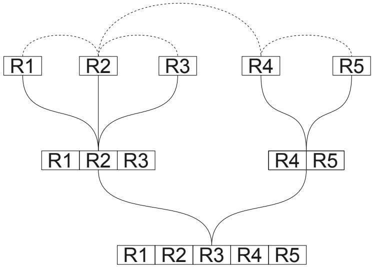
- Each rule can be represented by a condition graph.

$$R1(a > 17, d(X)),$$
$$R2(d(X), e(Y), g(Z)),$$
$$R3(c = on, g(Z)),$$
$$R4(e(Y), f(W)),$$
$$R5(b = Friday, f(W))$$



Gator – details

- Gator network structure:
  1. Alpha-memory.
  2. Optional beta-memory.
  3. P-node.
- Network optimisation:
  1. Connectivity heuristic.
  2. Disjointness constraint.
  3. Lowest-cost heuristic.



Inference algorithms – comparison

TREAT vs. RETE

- TREAT does not use beta-memory:
  - It saves time.
  - No redundant information – saves space.
- TREAT works directly on the conflict set.

Gator vs. TREAT

- RETE and TREAT are special cases of Gator.
- Gator beta-memory is optional and can have multiple inputs.
- Gator networks are always optimal according to certain cost functions.

Knowledge modularisation

Main goals of modularisation

1. to help manage large sets of rules,
2. to improve performance of inference algorithms,
3. to provide visualisation of the knowledge base,
4. to improve inference control strategies.

Note

There aren't any tools supporting all four goals.

CLIPS modules

- Rules in CLIPS may be grouped into modules.
- A module is a set of non–related rules.
- Each module has its own discrimination network.
- Inference algorithms test and fire only the rules in the module that has *focus*.
- When fired, any rule can change the current *focus*:

```
  ...
CLIPS> (defmodule A)
CLIPS> (defmodule B)
CLIPS> (defrule A::foo => (focus B))
CLIPS> (defrule B::bar => (return))
```

Hello World - CLIPS

```
(defmodule salutation
    (import MAIN ?ALL)
    (export ?ALL))
(defmodule greeting
    (import MAIN ?ALL)
    (export ?ALL))
(defmodule userMessage
    (import MAIN ?ALL)
    (import salutation ?ALL)
    (import greeting ?ALL))
```

```
(defrule MAIN::startrule ""
   =>
   (printout t "---- STARTING ----" crlf)
   (focus salutation greeting userMessage)
)

(defrule salutation::rule1 ""
   (gender female)
   (marital_status married)
   (age ?years)
   (test (>= ?years 10))
   =>
  (assert (salut Mrs.))
  (printout t "rule1 -> fired" crlf)
  (pop-focus)
)
```

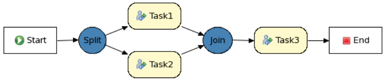
JESS modules

- JESS also supports rule grouping.
- Each module can contain any rules.
- A given rule always belongs to only one module.
- Inference algorithms always test all the rules.
- Only rules that are in the module in focus can be fired.
- When fired, any rule can change the current *focus*:

```
  ...
JESS> (defmodule A)
JESS> (defmodule B)
JESS> (defrule A::foo => (focus B))
JESS> (defrule bar => (return))
```

Drools Flow

- Provides visualisation and a graphical user interface (GUI):



- Rules are stored in one global knowledge base.
- Rules can be grouped into *ruleflow-groups*:

```
rule "Rule1"
ruleflow-group "Task1"
when
...
then
...
end
```

- Only rules from the current *ruleflow-group* are evaluated and fired.
- *Ruleflow-groups* determine the order of the rules evaluation and execution.

Hello World - DroolsExpert

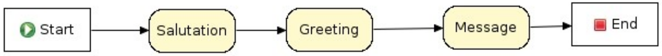
```
rule "Morning"
  ruleflow-group "Greeting"
  no-loop
  when
    d : DayTime(hour < 12);
  then
    d.setGreeting("Good morning");
    update( d );
end
rule "Afternoon"
  ruleflow-group "Greeting"
  no-loop
  when
    d : DayTime(hour >= 12, hour < 18);
  then
    d.setGreeting("Good afternoon");
    update( d );
end
```

```
rule "Salutation Mr"
  ruleflow-group "Salutation"
  no-loop
  when
    p : Person( age >= 10, male == true );
  then
    p.setSalutation("Mr.");
    update( p );
end

rule "UserMessage"
  ruleflow-group "Message"
  when
    p : Person( s : salutation);
    d : DayTime( g : greeting);
  then
  System.out.println(s + " " + g);
end
```

Hello World - DroolsFlow

```
rule "Salutation Mr"
  ruleflow-group "Salutation"
  no-loop
  when
    p : Person( age >= 10, male == true );
  then
    p.setSalutation("Mr.");
    update( p );
end
```

```
rule "Morning"
  ruleflow-group "Greeting"
  no-loop
  when
    d : DayTime(hour < 12);
  then
    d.setGreeting("Good morning");
    update( d );
end
```



```
rule "UserMessage"
  ruleflow-group "Message"
  when
    p : Person( s : salutation);
    d : DayTime( g : greeting);
  then
  System.out.println(s + " " + g);
end
```

Comparison

- All introduced shells support rule modularisation.
- Shells support modularisation on different levels:
  - CLIPS, Drools – only rules from the current module are evaluated.
  - JESS – only rules from the current module can be fired.
- Rule modularisation is not related to rule context.

Feature	CLIPS	Jess	Drools
Knowledge modularisation	Yes	Partiall	Yes
Knowledge visualisation	No	No	Yes
Formal rules representation	No	No	No
Knowledge base verification	No	No	No
Inferences strategies	DDI	DDI, GDI,	DDI
Inference algorithm	Rete	Rete	Rete
Allows for modelling dynamic processes	No	No	Yes

CLIPS applications

Constraint Programming within CLIPS

A generic knowledge base which enables CLIPS to solve CSPs (time table, cross-words, graph colouring, job shop scheduling, SAT problems)

Expert Surgical Assistant

A prototype surgery interface that allows surgeons to interact with virtual tissue and organ models using an intuitive combination of voice and gesture and also monitors their actions to give automatic feedback

Expert System Benchmarks

CLIPS versions of ARP (Aeronautical Route Planner), Waltz (Diagram Labeling), and Weaver (VLSI routing for channels and boxes) and other.

On-Line Nuclear Power Plant

A Java applet that simulates processes inside a nuclear power plant

Drools users

Sabre Holdings ™

DLR Deutsches Zentrum für Luft- und Raumfahrt e.V. in der Helmholtz-Gemeinschaft

DeCare℠ Systems Ireland, Ltd

eas enterprise architecture solutions

INNOVISION CORPORATION

chooz-it find & book online

actis.bsp Germany

Kiala

Auster

betfair sports casino poker

The End

Thank you for your attention!
Any questions?

Web Page: `http://geist.agh.edu.pl`

Powered by $\LaTeX$