Rule-based Systems

Uwe Egly

Vienna University of Technology
Institute of Information Systems
Knowledge-Based Systems Group

Overview

- Knowledge is often formulated in if-then manner
- We often have to deal with situations like

  if Cond1 $\wedge \ldots \wedge$ Condn **then** perform some actions
- Such a statement is called production rule or simply rule
- Often given by domain experts during knowledge acquisition
- In this lecture, we deal with rule-based systems (RBSs)
  - Presentation of the fundamentals of RBSs
  - Examples for such systems used (not only) for KBSs
  - Brief description is based on JBoss Drools
- Next lecture: Example of a rule-based system in geodesy

Why Rules?

- Rules are easier to read than source code
- Non-programmers (esp. domain experts) can write rules
- Rules are better for describing complex systems (because rules are declarative)
- Declarative means to describe what to do, not how to do it
- Rules are often independent from each other
- Rules have no predefined sequence for the application (they do not describe the control)

What are Rules and Rule-based Systems?

- Rules may express different types of reasoning:

premise	$\longrightarrow$	conclusion	logical implication
antecedence	$\longrightarrow$	consequence	infer from given precondition
evidence	$\longrightarrow$	hypothesis	interpretation of facts
situation	$\longrightarrow$	action	situated behavior
IF	$\longrightarrow$	THEN	informal paraphrases
left-side	$\longrightarrow$	right-side	can mean anything

- Well-known successful system include
  ILOG, JBoss Drools, CLIPS, Ruby ROOLS, NOBRE
- Historically: Rules used for XPS (e.g., DEC's XCon et al.)
- Today: "Business Rules", in the Semantic Web or Games

Expert Systems @ DEC (1)

Initial situation (around 1975)

- VAX computers were sold especially tailored for each client
- Need for 1000 technical editors (TEs)
  (=expert for configuring computer systems)
- No chance to hire them or to educate them
- Idea: Write a support program to boost efficency of TEs
- The program failed because
  - the problem complexity was too high
  - there were nondeterministic solutions
  - the configuration data changed too quickly
- How to overcome these difficulties?

Expert Systems @ DEC (2)

The solution

- Develop an expert system (named R1 and later XCon)
  - Developed together with CMU
  - Rule-based (final version had approximately 5000 rules)
  - Configured VAX computers from customer orders
  - Success rate 99%
- Success of XCon yielded the development of further XPSs
- J. McDermott received AAAI Classic Paper Award 1999 for

  R1: An Expert in the Computer Systems Domain

What is a Rule-based System?

- RBSs consists of the following parts
  - A collection of facts
    (short term knowledge of the KB, often case-specific)
  - A collection of rules (i.e., one or more rule bases)
    (long term knowledge of the KB, often domain-specific)
  - An inference engine
- Knowledge representation and reasoning are separated
- Two principle tasks:
  - Derive new facts
  - Determine whether a specific fact can be derived with the given rules and already known facts

Control Regimes for RBSs

- Two principle tasks imply two control regimes:
  - Forward chaining (data driven):
    start with facts, determine applicable rules, and apply one
  - Backward chaining (goal oriented):
    look for rules which decompose goal; solve smaller goals
- Is one better than the other?
  No general answer possible (depends on the application)
- We focus on forward chaining systems here

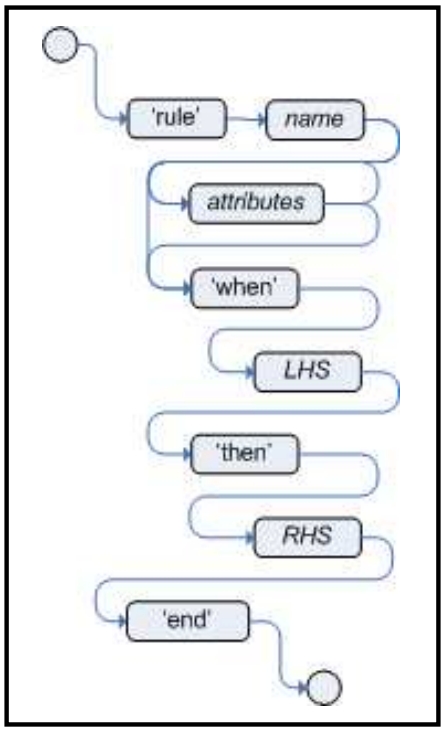
The Working Memory

- Place where the facts (objects relevant for rules) are stored
- Consists of Agenda (= Conflict Set), Truth Maintenance System , WM Event Support, etc.
- Some operations on the WM are (there are much more):
  - insert: put a new fact into the WM
  - retract: delete a fact from the WM
  - update: update a fact already in the WM
  - fireAllRules: find applicable rules and fire one of them
- Example:

```
Cheese brie = new Cheese("brie");
FactHandle bHandle = session.insert(brie);
```

The Rules

- Rules have the form shown on the right
- Each rule typically stored in own file
- Rules cannot be called directly
- Most important attribute:
  - salience: Rule priority as an int
- LHS: Can be highly complex (exa below)
- RHS:
  - Small code part (usually insert, retract, update WM data)
  - No complicated program structure
  - Keep RHS simple and readable

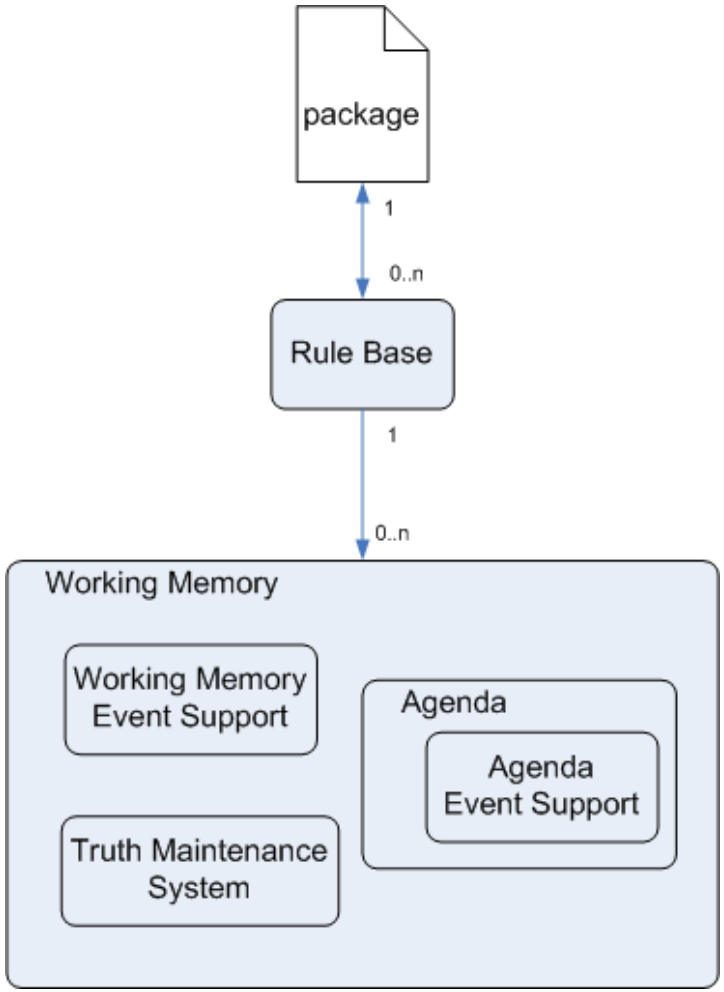


Examples for LHSs

- Cheese type is "brie" or price < 10, and age is mature
  ```
  Cheese(type=="brie" || price<10, age=="mature")
  ```
- Check for earlier
  ```
  Cheese(bestBefore < "27-Oct-2008")
  ```
- Return Value Restriction: girlAge takes age from first
  ```
  Person(girlAge :  age, sex == "F")
  Person(age == (girlAge + 2)), sex == 'M')
  ```
- Positive conditions check for existence of something in WM
- not checks for non-existence of something in the WM
  ```
  not (Bus(color=="red") and Bus(color=="blue"))
  ```
- With not, nonmonotonic behavior comes into play

Rule Bases

- Contains the rules (usually ready to run, i.e., compiled)
- Contains the WMs
- Initializes WMs (initial facts)
- Usually contains parts of the inference engine
- Usually highly configurable



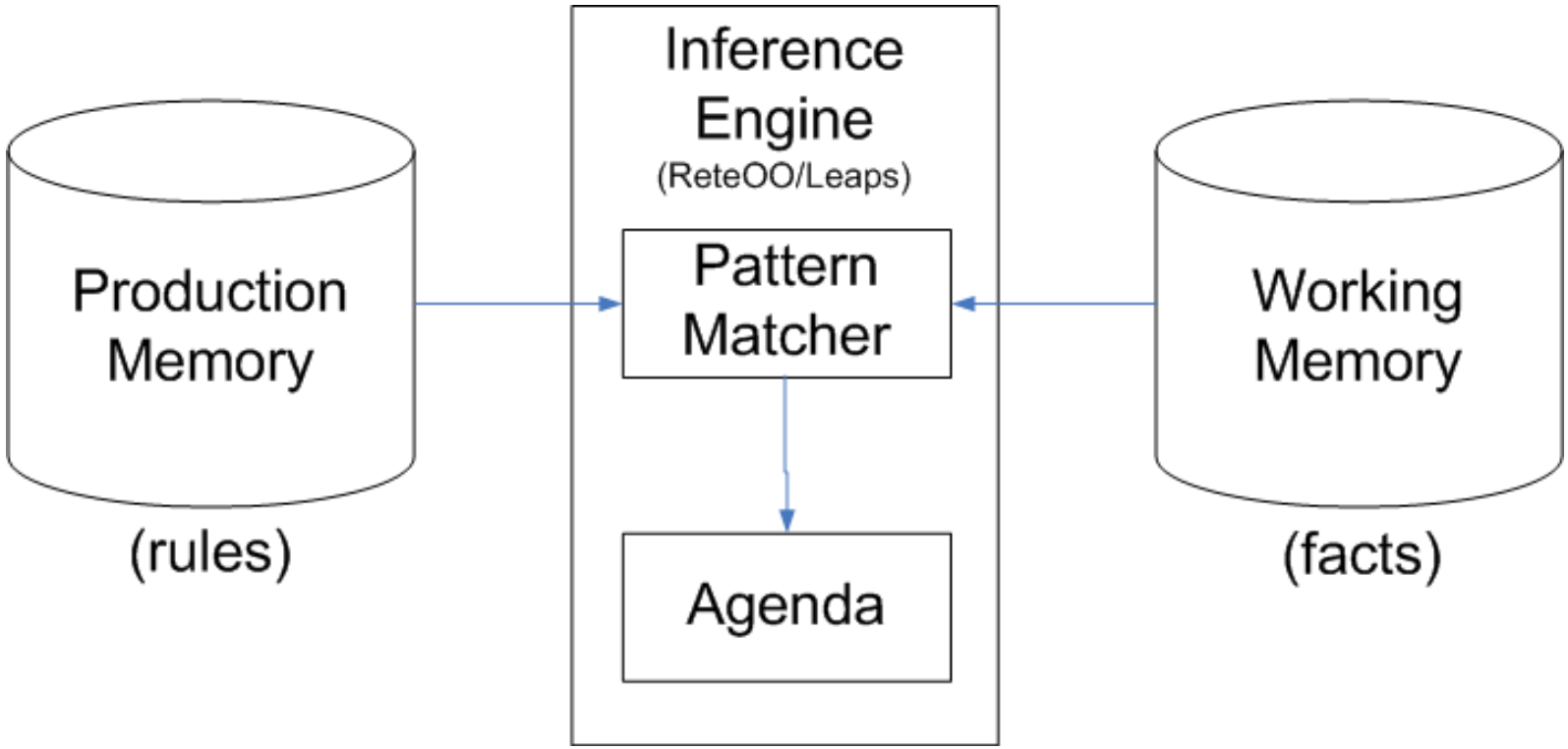
The Inference Engine

- Matches facts in the WM against rules (= productions)
- Is able to scale to a large number of rules and facts
- Matching determines the applicable relevant knowledge in the given situation
- Matching of many rules against many facts computationally expensive (use special algorithms like Rete, ReteOO, etc.)
- Matching often yields $>$ 1 applicable rule (put in Agenda) (these rule instances are said to be in conflict)
- Use conflict resolution to pick one for firing
- Firing: executing the RHS of an applicable rule instance

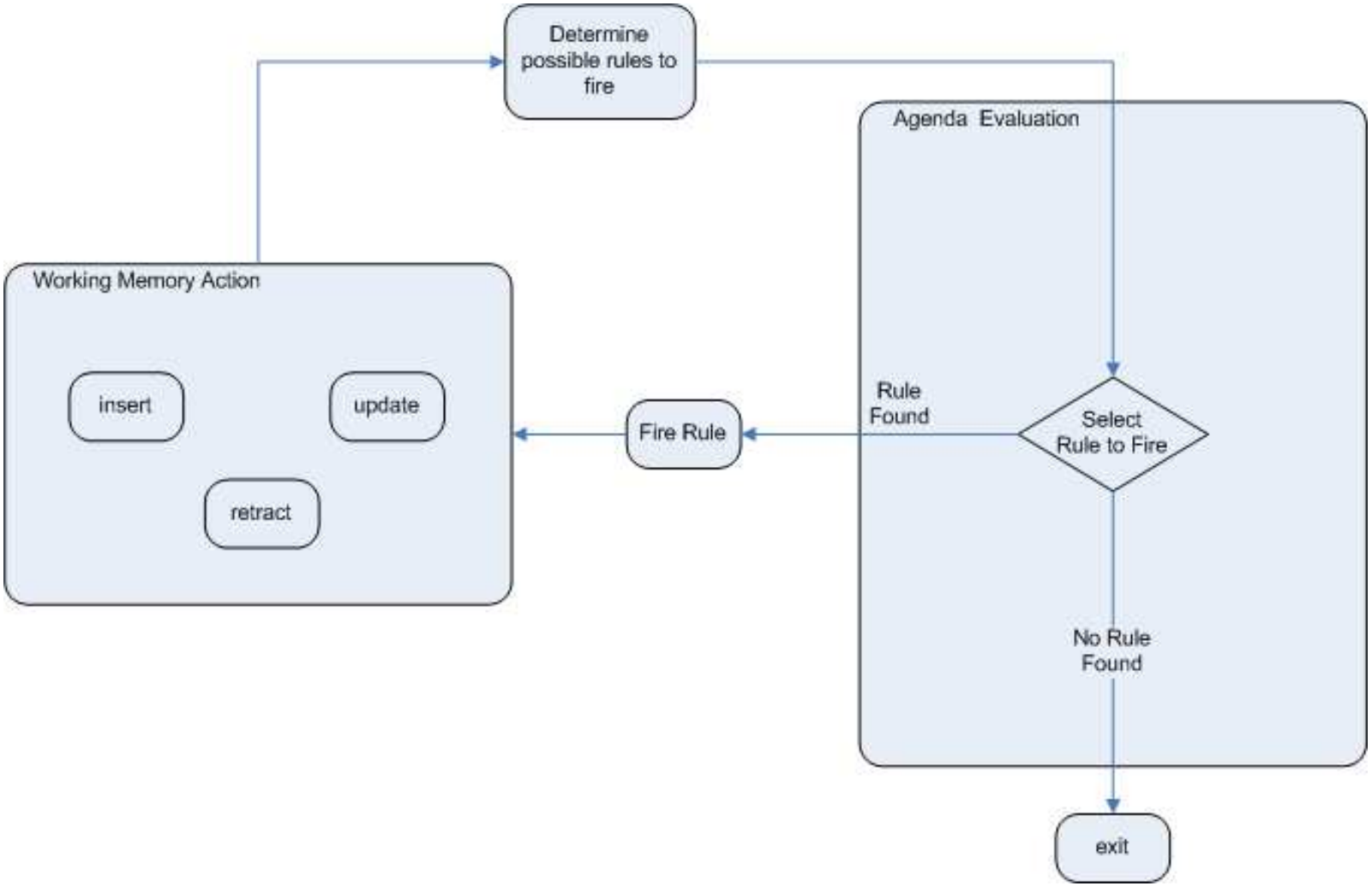
What is a Rule Instance?

- Rule instance consists of
  - a reference to a rule and
  - list of references to objects in WM satisfying the pos conds (positive means no not at the beginning)
- Each reference is a witness that corresponding cond is true (I.e., there is an object in the WM which satisfies the condition)
- Clearly, no reference for the negative conditions (why?)

An Architectural Overview of the Inference Engine



The Recognize-Act-Cycle



Conflict Resolution (CR)

- It is required if multiple rule instances are on the agenda
- Do not count on the rules firing in any particular order
  (In general, we represent in a declarative way!)
- Sometimes. declarative way violated for efficency
- Standard custom conflict resolution strategies are often
  - Salience (=rule priority given by the knowledge engineer)
  - LIFO
- Custom CR strategies possible and may be based on:
  - Specificity:
    Rules with more specific conditions in the LHS are preferred
  - Rule instances with newer information are preferred
  - Rule instances with rules recently fired are preferred
  - Or simply choose randomly

An Introductory Example

Assume that we have `Items` which are strings and `Customers` who have a `cart` (array of items)

```
Customer customer = new Customer("Fred Flinstone");
customer.addItem(new Item("brie"));
customer.addItem(new Item("cheddar"));
customer.addItem(new Item("feta"));
workingMemory.insert(customer);

rule "Msg to customers who haven't bought any brie"
when
  $c :  Customer($cart :  cart ->
    (!$cart.includes(new Item("brie"))))
then
  $c.sendMessage("Brie is your best friend");
end
```

A More Complicated Example: The Age Problem

An old man (O) asks a mathematician (M) to guess the ages of his three sons. Listen to their conversation:

O : The product of their ages is 36.

M : I need more information.

O : Over there you can see a building. The sum of their ages equals the number of the windows in that building.

M : I need more information.

O : The eldest son has blue eyes.

M : I got it.

What are the ages of the three sons of the old man? And how many windows does the building have? Solve the problem with a rule-based approach!

Analysis of the Age Problem

- This problem was the first one of the Drools contest
  (a similar one occurs in How to Solve It: Modern Heuristics by Michalewicz and Fogel)
- We discuss the solution of Elmo Nazareno
  (http://ningning.org/blog2/?p=120)
- First grasp and formalize info given by O (next slides)

Analysis of the Age Problem (cont'd)

- a_1, a_2, a_3: the age of the youngest, middle, eldest son
- The product of their ages is 36: $a_1 \cdot a_2 \cdot a_3 = 36$
- How many such products are possible?

a_1	a_2	a_3
1	1	36
1	2	18
1	3	12
1	4	9
1	6	6
2	2	9
2	3	6
3	3	4

- We do not know the ages; can be every possibility

Analysis of the Age Problem (cont'd)

- (M) knows the number of windows, but we do not!

a_1	a_2	a_3	sum
1	1	36	38
1	2	18	21
1	3	12	16
1	4	9	14
1	6	6	13
2	2	9	13
2	3	6	11
3	3	4	10

- The solution must be one of the indicated blue lines since otherwise (M) would have the solution already
- The eldest son has blue eyes; therefore $a_1 \leq a_2 < a_3$
- How can we use this info to come up with facts and rules?

Analysis of the Age Problem (cont'd)

- (M) knows the number of windows, but we do not!

a_1	a_2	a_3	sum
1	1	36	38
1	2	18	21
1	3	12	16
1	4	9	14
1	6	6	13
2	2	9	13
2	3	6	11
3	3	4	10

- The solution must be one of the indicated blue lines since otherwise (M) would have the solution already
- The eldest son has blue eyes; therefore $a_1 \leq a_2 < a_3$
- How can we use this info to come up with facts and rules?

A Solution of the Age Problem: The Facts

- Facts are instances of objects with an attribute `age`
- Generate the facts Son with age *i*

```
for (int i = 1; i <=36; i++)
  if ((36 % i) == 0)
     workingMemory.insert(new Son(i));
```

A Solution of the Age Problem: The Rule (1)

Basic idea: Search for the two different ordered sequences with identical sums and choose the good sequence

- Find two different ordered sequences of three ages
- The product of each sequence must equal 36
- The sums of the two sequences must be equal
- The eldest must not have a twin

A Solution of the Age Problem: The Rule (2)

```
rule "determine ages"
when
  Son($a3:  age)
  Son($a2:  age < $a3)
  Son($a1:  age <= $a2)

  Son($w3:  age)
  Son($w2:  age <= $w3)
  Son($w1:  age <= $w2)

  eval($a1!=$w1 && $a2!=$w2 && $a3!=$w3)
  eval(($a1 * $a2 * $a3) == 36)
  eval(($w1 * $w2 * $w3) == 36)
  eval(($a1 + $a2 + $a3) == ($w1 + $w2 + $w3))
then
  System.out.println("eldest:  " + $a3 +
    " middle:  " + $a2 + " youngest:  " + $a1 );
end
```

Solution of the Age Problem

- Try to figure out a solution and . . .
- answer the two questions mentioned before
- You may want to use Drools; the next slide gives a rough impression of the integration into Eclipse and some corresponding tools.
- Infos can be found using the link `http://labs.jboss.com/drools/`
- Next lecture: a report of a KBS in geology
  (to detect/classify mass movements like land slides)

Screenshot Drools in Eclipse with Rete Viewer