ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Rule-based systems and Forward Reasoning

Prof. Ing. Federico Chesani

DISI

Department of Informatics – Science and Engineering

Notice

These slides are largely an adaptation of existing material, including slides by Dr. Davide Sottara and Dr. Stefano Bragaglia. I am especially grateful to both for letting me access and use their own material.

Sharing: a copy of these slides can be downloaded from the servers of the University of Bologna and stored for personal use only. Please do not redistribute.

Why rules?

- Rules are a very common way for eliciting knowledge, and how decisions should be in **given** and **known contexts.**
- Rules are easy to understand
- Rules are easy to be expressed and elicitated
- Rules are very similar to several high-level cognitive tasks: humans adopt rules in several situations (default rules sometimes are applied at an unconsciousness level)
- Rules have a solid formal background in logic, and have been investigated since classical philosophy

- Backward Chaining (e.g., Prolog)
- Forward Chaining, such as production rules (e.g., RedHat Drools)

Rule-based frameworks and process automation

- All the big ICT vendors provide suites for the business process automation...
- ... and each suite contains also a rule-based language/engine.

- **Why?** The idea is that process managers can directly define (executable) rules, so that lager parts of business process can be automated.

- Microsoft: BizTalk Framework, and the Business Rules Engine https://docs.microsoft.com/en-us/biztalk/core/business-rules-engine
- IBM: Operational Decision Manager https://www.ibm.com/automation/software/business-rules-management
- Oracle: JBoss Enterprise Application Platform and Drools https://www.drools.org/
- OpenRules http://openrules.com/

The Decision Model and Notation: a curious case

- Few years ago, a novel research area emerged, namely the Business Process Management research field
- In few years it became a large community with a strong industry participation

- Initially, the focus was on how to describe business process…
- … until the BPMN standard has been proposed by OMG, and widely accepted

The Decision Model and Notation: a curious case

- After the definition of the BPMN standard, in the BPM community they soon realized that decisions were a fundamental part of any business process

- How decision are taken in the business?
- Which criteria?
- How to describe these criteria?

The answer was again RULES!!! But in a "nicer" form…

- The DMN notation was proposed…
- …a simplified version of forward rules!!!

How are rules made?

The simplest form of a rule is the logical implication:

$$p_1, \ldots, p_i \rightarrow q_1, \ldots, q_j$$

- $p_1, \ldots, p_i$ are called the **premises**, **antecedents**, the **body** of the rule, or the **Left Hand Side** (**LHS**)
- $q_1, \ldots, q_j$ are called the **consequences**, the **consequent,** the **head** of the rule, or the **Right Hand Side** (**RHS**)

Examples:

- If it rains, then the road will become wet and slippery
- If you don't study, you will fail the exam
- If don't pay the tax within the deadline, you will be fined for 15% of the due amount, plus 4.15€ for each day of delay in the payment

How to reason with rules?

Reasoning with rules has been systematized in classic greek philosophy (Aristotle, syllogism). The default reasoning mechanism is **Modus Ponens**:

Peirce Notation:

$$\frac{p_1, \ldots, p_i \qquad p_1, \ldots, p_i \rightarrow q_1, \ldots, q_j}{q_1, \ldots, q_j}$$

- We know for sure that $p_1, \ldots, p_i$ are true
- We know for sure that the rule holds

- We derive the truthness of $q_1, \ldots, q_j$

- NOTICE: this is not the only possible inference mechanism…

From static knowledge bases to dynamic ones: the "Production Rules" approach

- In previous examples we have seen backward chaining applied to a static knowledge base
- The same can be said for several forward chaining approaches:
  - the knowledge base is defined before the reasoning step;
  - the consequences are derived during the reasoning step;
  - the knowledge base is augmented with only facts that have been the result of the application of the Modus Ponens rule to facts and rules in the knowledge base

- What if, during the reasoning step, new information/new facts become available?

- A solution (naive): **when needed**, restart the reasoning from scratch...
  - Which cost?
  - When it is worthy to restart the reasoning?

From static knowledge bases to dynamic ones: the "Production Rules" approach

- In the production rules approach, **new facts can be dynamically added to the knowledge base**
  - Which difference with the lemma generator seen in prolog meta-interpreters?

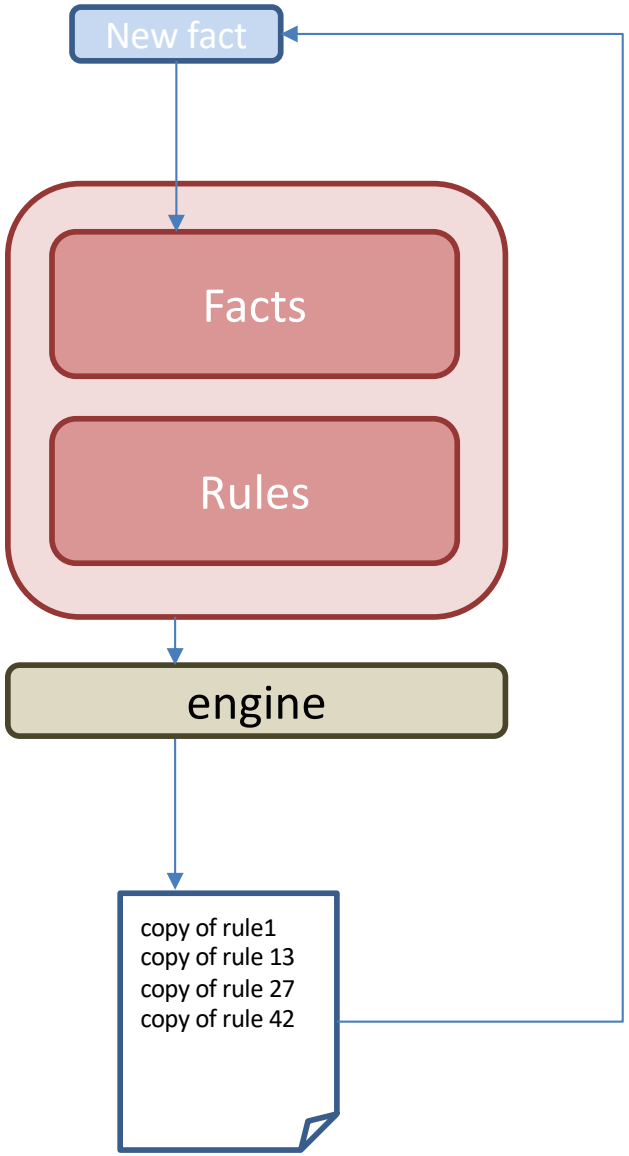
- if new facts matches with the left hand side of any rule, the reasoning mechanism is **triggered**, until it reaches **quiescence** again...

From static knowledge bases to dynamic ones: the "Production Rules" approach

- Production Rule systems: they allow to explicitly have **side effects** in the RHS
  - side effects on the external world
  - side effects on the knowledge base
- Which side effects on the working memory?
  - **Insertion** of facts
  - **Deletion** of facts
  - .., and consequent **retriggering** of rules

- we are leaving the "logical" setting, heading towards a more "procedural" framework
  - what if new facts are **inconsistent** with the existing ones?
  - what if new facts **trigger more rules**? which one to start first?
  - what if new facts trigger a rule that generates **side effects**?
  - what if a new fact trigger a rule, and that rule **deletes a fact** that was the antecedent of a rule already triggered?

Production Rules



Steps

When a new fact is **added** to the knowledge base, or **initially**:

1. **Match**: search for the rules whose LHS matches the fact and decide which ones will trigger.
2. **Conflict Resolution/Activation**: triggered rules are put into the Agenda, and possible conflicts are solved (FIFO, Salience, etc).
3. **Execution**: RHS are executed, effects are performed, KB is modified, etc.
4. Go to step1

Working memory

- Data structure at the base of Production Rules Systems.
- It contains:
  - the current set of facts (initially, it is just a copy of the KB)
  - the set of rules
  - the set of the (copies of) "partially matched" rules

- Performances of PRS might depend on the efficiency of the WM...

The RETE Algorithm

- Charles Forgy, “Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem”, Artificial Intelligence, 19, pp 17-37, 1982

- **Focuses on the step 1: "Match"**
- The match step consists on computing which are the rules whose LHS is "matched"
- LHS is usually expressed as a conjunction of patterns
- Deciding if the LHS of a rule is satisfied/is matched, consists on verifying if all the patterns do have a corresponding element (a fact) in the working memory
- It is a "many patterns" vs "many objects" pattern match problem

The RETE Algorithm

Example

- Rule1: "when a Person p is added to the WM, and she is a student, send her a greeting message"
- Rule2: " when a Person p is added to the WM, and she is a professor, send her a reminder about the slides"

- Fact: "a Person whose name is Sara, and whose role is student, is added to the WM"

- The problem is how to determine efficiently which are the rules that should be triggered…
- … in this example, the cost is linear with the number of rules
- … iteration over the rules

The RETE Algorithm

Another example:

- Rule1: "when a Person p is added to the WM, and she is a student, and for every other Person stored in the WM, that are student themselves, send a message to the latter about the newcomer."

- Fact: "a Person whose name is Sara, and whose role is student, is added to the WM"

- The problem, again, is how to determine efficiently which are the rules that should be triggered…
- … in this example, if we have to evaluate the LHS at the insertion (from scratch) the cost is linear with the dimension of the facts
- … iteration over the facts

The RETE Algorithm – avoid iteration over facts

- RETE focuses on determining the so called "conflict set"…
- … i.e., the set of all possible instantiations of production rules such that

  <Rule (RHS), List of elements matched by its LHS>

Idea: avoid iteration over facts by storing, for each pattern, which are the facts that match it

- When a new fact is added, all the patterns are confronted, and the list of matches is updated
- When a fact is deleted, the patterns are updated as well
- When a fact is modified…

The RETE Algorithm – avoid iteration over the rules

Idea: "compile" a LHS into a network of nodes.

(At least) Two types of patterns:

1. Patterns testing intra-elements features
2. Patterns testing inter-elements features

Example: "When a student whose name is X, when a professor with name X, notify a warning to the deputy head of the school".

The RETE Algorithm – avoid iteration over the rules

Idea: "compile" a LHS into a network of nodes.

(At least) Two types of patterns:

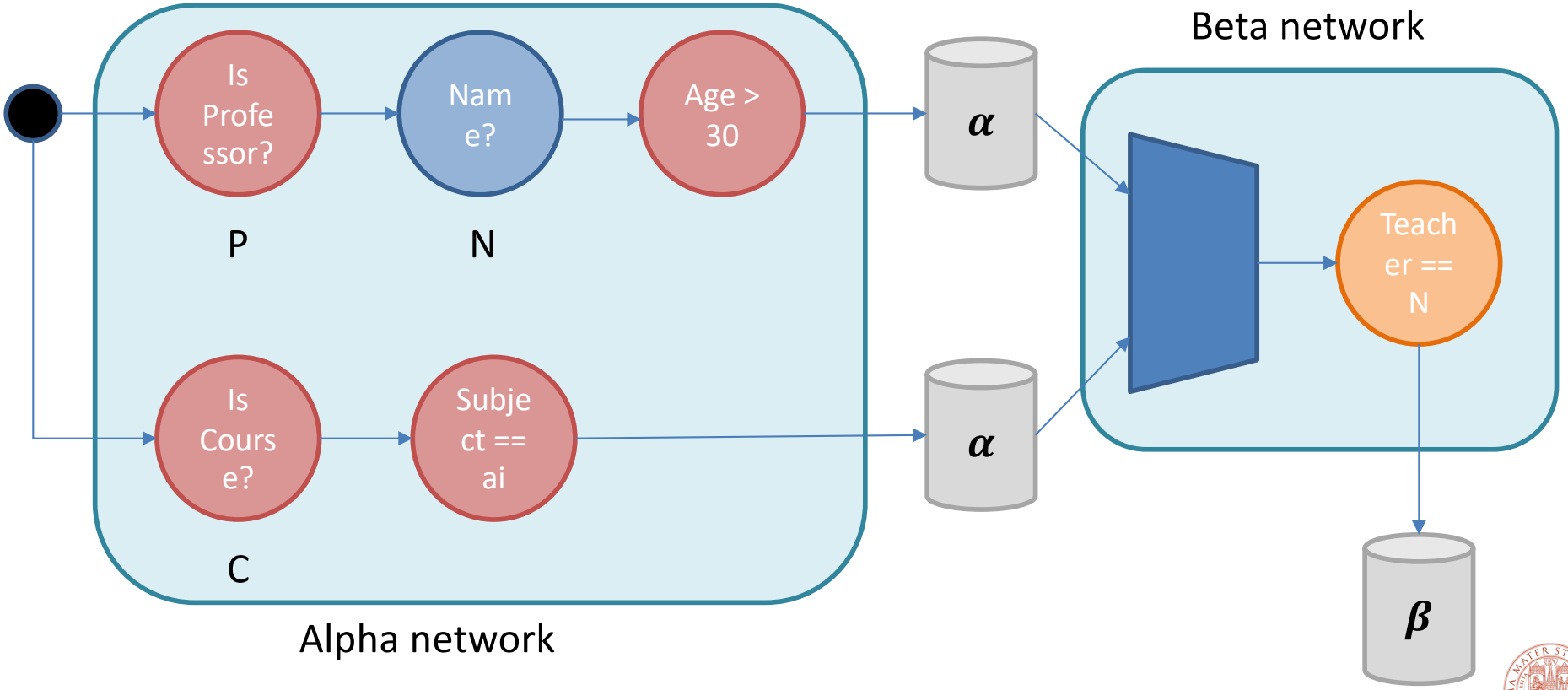
1. Patterns testing intra-elements features
2. Patterns testing inter-elements features

- Intra-elements patterns are compiled into alpha networks, and their outcomes are stored into alpha-memories

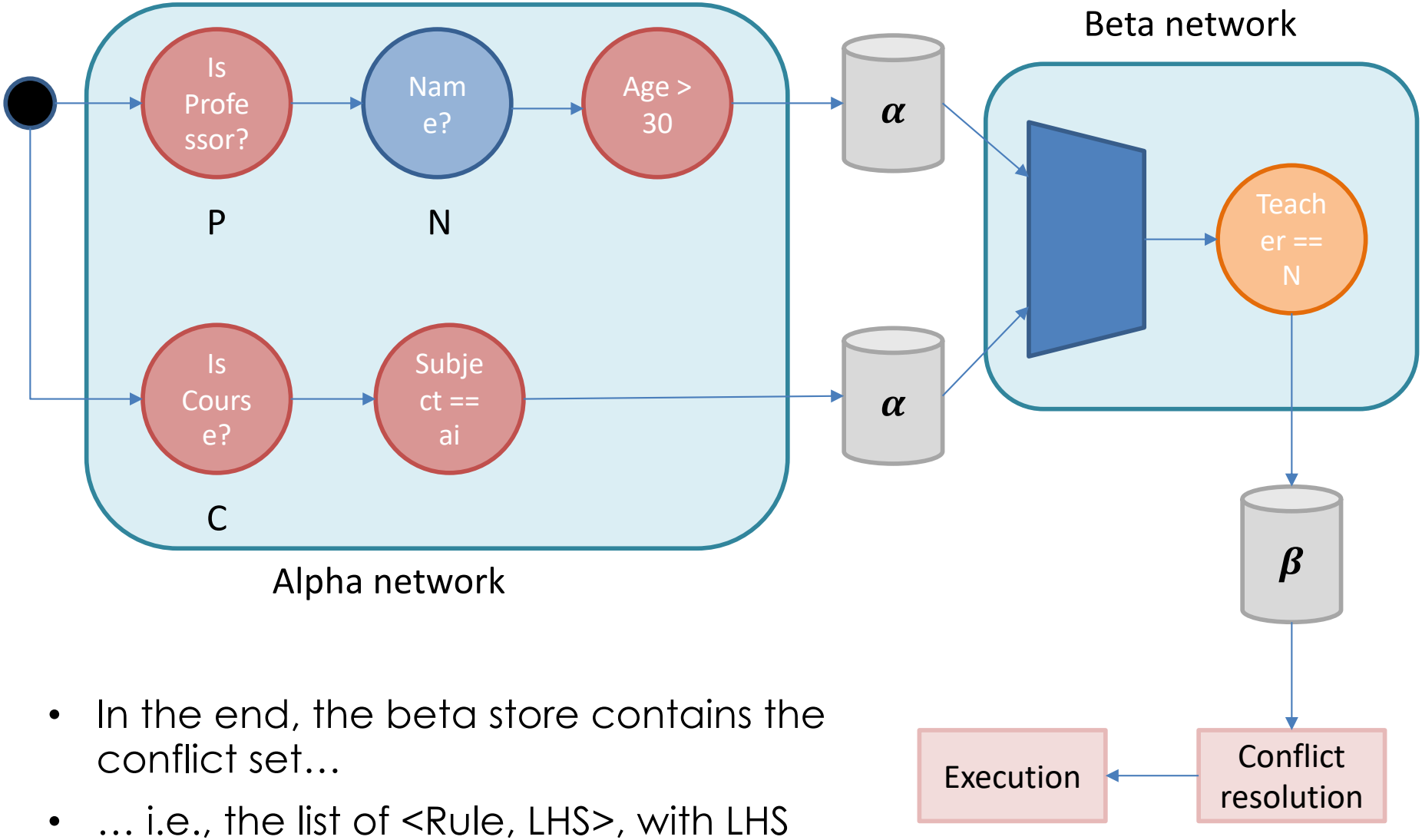
- Inter-elements are compiled into beta-networks, and their outcomes are stored into beta-memories

Rete algorithm – an example

When a Professor P with name N and age>30, when a Course C with subject "ai" and teacher N THEN do something



Rete algorithm – an example



- In the end, the beta store contains the conflict set…
- … i.e., the list of <Rule, LHS>, with LHS completely matched, and ready to be executed.

… and the other steps?

- **Step2 conflict resolution**: given that many rules can be fired/executed in their RHS, which one will be executed first? In which order will they be executed?
  - rule priority, aka salience
  - rule order
  - some temporal attribute (e.g., the time at which facts were inserted into the WM)
  - complexity of a rule
  - …

… and the other steps?

- **Step3 execution**: each activated instance of a rule will be executed

- By default, all the rules are executed in a cycle. Possible WM modifications will be tackled in another cycle…
- … however, it is often possible to modify this behaviour

- What about possible loops?

The DROOLS Framework

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

The Drools Framework

- Supported by JBoss/RedHat
- It comes with a plethora of tools for supporting the definition the business logic. Using JBoss terminology, Drools is a "Business Logic Integration Platform".

- Drools Expert – the Rule Engine itself
- Drools Fusion – support for the Event Processing
- jBPM – workflow engine
- OptaPlanner – Automated planning

The Drools Framework

- From the Drools manual:

- *"Drools was also born as a rules engine several years ago, but following the vision of becoming a single platform for behavioral modelling, it soon realized that it could only achieve this goal by crediting the same importance to the three complementary business modelling techniques:*

1. *Business Rules Management*
2. *Business Processes Management*
3. *Complex Event Processing"*

Drools Expert

Few characteristics:

- Forward reasoning approach
- **Based on the RETE algorithm, and current version on Phreak algorithm (an evolution of RETE and ReteOO)**

- Open Source
- Java-based
- The language is proprietary, but easy to extend it with custom operators, etc. (Java...)
- Provides an IDE (Eclipse), but also web interfaces
- Requires only a superficial comprehension of the Java language
- Open community of developers, very keen to provide quick answers on blogs and relay chats

The Drools language

- Rules have the structure:

```
rule "Any string as id of the rule"
     // possible rule attributes
when
     // LHS: premises or antecedents
then
     // RHS: consequents or conclusions...
side effects
end
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Left Hand Side – LHS

It is a **conjunction** of (disjunctions of) patterns.

- A **pattern** is the atomic element for describing a conjunct in the LHS.
- describes a **fact** (that could appear in the WM)
- describes also the **conditions** about the specific fact
- Indeed, it is a sort of a filter for the objects/facts within the WM

- Example: "When a Person is inserted in the system, send him a greeting email"

```
rule "Greeting email"
when
        Person( )
then
        sendEmail("greetings")
end
```

Left Hand Side – LHS

Example: "When a Person is inserted in the system, send him a greeting email"

```
rule "Greeting email"
when
        Person( )
then
        sendEmail("greetings")
end
```

- The Person ( ) pattern is a base one, and corresponds to the constraint:
- **`x instanceof Person ?`**

- Notice that Drools is Java-based, hence it inherits the instanceof definition from Java (e.g., no multiple inheritance).

Left Hand Side – LHS

Example: "When a Person is inserted in the system, send him a greeting email"

```
rule "Greeting email"
when
        Person( )
then
        sendEmail("greetings")
end
```

When the rule above is triggered?

1. If the engine is already running, as soon as a fact Person() is inserted in the working memory;
2. At the start of the rule engine, if a fact Person() was already in the working memory (initial setup of the WM).

Left Hand Side – LHS

The basic pattern alone is poor (from the expressiveness viewpoint)…

… support to constraints on the object fields (**field constraints**):

- classical operators, extended to primitive types and String
  - ==, <, >=, …
- connected through:
  - logical and: ‘ , ’ or ‘ &&’
  - logical or: ‘ || ‘

Example: ”When a Person of age, whose name is “federico”, is inserted in the system, send him a greeting email”

```
rule "Greeting email"
when
        Person( name=="Federico", age >= 18)
then
        sendEmail("greetings")
end
```

Left Hand Side – LHS

Example: "When a Person of age, whose name is "federico", is inserted in the system, send him a greeting email"

```
rule "Greeting email"
when
        Person( name=="Federico", age >= 18)
then
        sendEmail("greetings")
end
```

Again, the meaning is inherited from Java…

```
x.getName().equals("Federico") && x.getAge() >= 18
```

What if name and age fields are not accessible through standard java bean notation?

COMPILE-TIME or RUN-TIME ERROR!

Left Hand Side – LHS

What if we need to keep in mind certain values/fields/objects?

Variables:

- any literal, starting with the ‘ $’ character
- assigned through the ‘ : ’

Variables can be assigned to both objects/facts, as well as fields (constrained or not)

Example: ”When a Person of age, whose name is “Federico”, is inserted in the system, send him a **personalized** greeting email”

```
rule ”Greeting email”
when
        $p : Person( $n : name, $n==”Federico”, age >= 18)
then
        sendEmail($p.getEmailAddr(), “greetings to ” + $n)
end
```

Left Hand Side – LHS

What if we want to trigger rules when more facts appear at the same time? **Join**: allows to specify multiple patterns in the LHS

- all the pattern must be intended in logical AND: the rule trigger when all the pattern are satisfied
- all the possible combinations are generated by using all the objects that match with the premises

Constraints between different patterns can be imposed by using the variables, whose visibility scope is the whole rule

Example: "When a married couple is inserted in the system, do …"

```
rule "Married couple"
when
        $p1 : Person( $n1 : name)
        $p2 : Person( $n2 : name, $p2.marriedWith() == $n1 )
then
        ...
end
```

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Left Hand Side – LHS

What if we want to trigger rules when more facts appear at the same time? **Join**: allows to specify multiple patterns in the LHS

- **all** the **possible combinations are generated** by using all the objects that match with the premises

Example: "When a married couple is inserted in the system, do …"

```
rule "Married couple"
when
        $p1 : Person( $n1 : name)
        $p2 : Person( $n2 : name, $p2.marriedWith() == $n1 )
then
        ...
end

// Person("Federico", marriedWith="Elena")
// Person("Elena", marriedWith="Federico")
```

Left Hand Side – LHS

What if we want to trigger rules when more facts appear at the same time? **Join**: allows to specify multiple patterns in the LHS

- Constraints between different patterns can be imposed by using the variables, whose visibility scope is the whole rule
- The keyword "this" can be used in field constraints to refer to the current object/fact

Example: "When a married couple is inserted in the system, do …"

```
rule "Married couple"
when
        $p1 : Person( $n1 : name)
        $p2 : Person( $n2 : name, this.marriedWith() == $n1 )
then
        ...
end
```

Left Hand Side – LHS

Quantifiers

- It is also possible to filter on the basis of three quantifiers:

- exists P(…) : there exists at least a fact P(…) in the working memory

- not P(…) : the working memory does not contain any fact P(…)
    - when is such test performed?

- forall P(…) : trigger the rule if all the instance of P(…) match

Left Hand Side – LHS

Quantifiers

It is also possible to filter on the basis of three quantifiers:

exists P(…) : there exists at least a fact P(…) in the working memory

Example: "Print the name of all those persons that have been fined at least once …"

```
rule "At least one fine"
when
        $p1 : Person( $n1 : name)
        exists Fine( subject == $p1)
then
        System.out.println($n1 + " was fined at least once");
end
```

- What if the person has received more fines?
- The rule triggers zero or one time.

Left Hand Side – LHS

Quantifiers

It is also possible to filter on the basis of three quantifiers:

- not P(…) : the working memory does not contain any fact P(…)

Example: "Print the name of all those persons that have never been fined"

```
rule "Never fined"
when
        $p1 : Person( $n1 : name)
        not Fine( subject == $p1)
then
        System.out.println($n1 + " was never fined");
end
```

Left Hand Side – LHS

Quantifiers

It is also possible to filter on the basis of three quantifiers:

- forall P(…) : trigger the rule if all the instance of P(…) match

Example: "Print the name of the persons that have been fined only for speed"

```
rule "Speed-only fined"
when
        $p1 : Person( $n1 : name)
        forall Fine( subject == $p1, reason="speed")
then
        System.out.println($n1 +" was fined for speed");
end
```

- The rule triggers zero or many time.

Left Hand Side – LHS

Facts… how are they made?

- LHS are defined through the use of patterns, that provide a way for expressing match constraints
- Facts are put into the WM during execution, or they are already there (initialization)

How facts are made?

a) Java-based Beans, import clause at the beginning of a rule file
   `import it.unibo.bbs.dss.Person;`

b) Directly defined within the rule file, with explicit list of fields (internally, mapped as Java Beans)

```
declare Person
        name : String
        age : int
end
```

Right Hand Side – RHS

Consequences can be of two types:

a) "Logic" consequences: they affect the working memory…

- Insert new facts into the WM (and possibly trigger rules)
- Retract existing facts
- Modify existing facts (and possibly re-trigger rules)

b) "Non-Logic" consequences:

- Any (external) side effect
- pieces of Java code that will be executed

Right Hand Side – RHS

Insert

- In the RHS it is possible to create new facts, and directly insert them into the WM.
- If the new fact matches with the LHS of any rule, then the rule will possibly trigger.

"When a registered person logins correctly, mark it as logged-in; when it logs out, check if it was logged in, and in positive case, say "goodbye" "

```
declare Logged
        person : Person
end
rule "log in"
when
        $p : Person ()
        $e : LogInEvent(person == $p)
then
        Logged ooo = new Logged();
        ooo.setPerson($p);
        insert(ooo);
end
```

```
rule "log out"
when
        $ooo : Logged ($p:person)
        $e : LogOutEvent(person == $p)
then
        System.out.println("Goodbye!");
end
```

Right Hand Side – RHS

Insert

In the RHS it is possible to create new facts, and directly insert them into the WM.

- If the new fact matches with the LHS of any rule, then the rule will possibly trigger.

"If you receive an email, immediately reply with "out-of-office" "

```
declare EMail
          sender : String
          dest : String
end

rule "automatic reply"
when
          $e : EMail ($s:sender, $d:dest)
then
          Email em = new Email();
          em.setSender($d);
          em.setDest($s);
          sendMail(em);
          insert(em);
end
```

Right Hand Side – RHS

InsertLogical

In the RHS it is possible to create new facts, and directly insert them into the WM.

- If the new fact matches with the LHS of any rule, then the rule will possibly trigger.
- the facts that were inserted are **automatically retracted** when the conditions in the rules that inserted the facts are no longer true.

TMS?

Right Hand Side – RHS

Retract

When desired, it is possible to remove facts from the WM.

- Rules partially instantiated are discarded coherently
- Side effects are not retracted (it couldn't be possible)

"When a registered person logins correctly, mark it as logged-in; when it logs out, check if it was logged in, and in positive case, say "goodbye" "

```
declare Logged
        person : Person
end

rule "log in"
when
        $p : Person ()
        $e : LogInEvent(person == $p)
then
        Logged ooo = new Logged();
        ooo.setPerson($p);
        insert(ooo);
end
```

```
rule "log out"
when
        $ooo : Logged ($p:person)
        $e : LogOutEvent(person == $p)
then
        retract($ooo);
        // delete($ooo);
        System.out.println("Goodbye!");
end
```

Right Hand Side – RHS

Modify/Update

A combination of retract and insert, applied consecutively...

- update: notifies the engine that an object has changed; changes can happen externally, or in the RHS part of the rule
- modify: takes all the modifications that should be applied, apply them, and notify the engine of the changes

"When a person logins, update the date of the last login "

```
declare LastLogged
        person : Person
        lastLogin : Date
end
rule "log in"
when
        $p : Person ()
        $e : LogInEvent(person == $p)
        $log : LastLogged(person == $p)
then
        $log.setLastLogin(new Date());
        update($log);
end
```

```
rule "log in"
when
        $p : Person ()
        $e : LogInEvent(person == $p)
        $log : LastLogged(person == $p)
then
        modify ($log) {
                setLastLogin(new Date())
        }
end
```

Right Hand Side – RHS

Insert, Modify/Update and loops

Insert and modify/update operations can be easily subject to 1-setp loops.

To avoid loops, there is the no-loop keyword:

```
rule "log in"
no-loop
when
        $p : Person ()
        $e : LogInEvent(person == $p)
        $log : LastLogged(person == $p)
then
        modify ($log) {
        setLastLogin(new Date())
        }
end
```

The intended meaning is that the rule shouldn't trigger, if the objects are modified by the rule itself…

Left Hand Side – LHS

Other features

It is also possible to filter on the basis of three quantifiers:

- from Collection<P(…)> : allows to evaluate LHS against a Collection of objects even if these objects are not in the WM
- collect(P(…)) : constructs a Collection of objects stored in the WM
- accumulate (P(…), aggregate_functions) : allows to collect a set of instances of P(…), and to extract aggregated information

```
rule "Customers age"
when
        accumulate ( Person( $a : age),
                        $max : max( $a ),
                        $min : min ( $a ),
                        $avg : average ($a )
        )
then
        …
end
```

Conflict resolution … salience

- Rules can have the property salience: the higher the value, the higher the priority of their execution

```
declare Logged
        person : Person
end

rule "log in"
        salience 100
when
        $p : Person ()
        $e : LogInEvent(person == $p)
then
        Logged ooo = new Logged();
        ooo.setPerson($p);
        insert(ooo);
end
```

Conflict resolution … agenda group

- Rules can have the property agenda-group: group of rules are selected to be executed from an external methods setFocus(…);

```
declare Logged
        person : Person
end

rule "log in"
        agenda-group "log"
when
        $p : Person ()
        $e : LogInEvent(person == $p)
then
        Logged ooo = new Logged();
        ooo.setPerson($p);
        insert(ooo);
end
```

Conflict resolution … activation group

Rules can have the property activation-group

- Among all the activated rules of the same group…
- … only one of them is executed…
- And the others are discarded.

```
declare Logged
        person : Person
end

rule "log in"
        activation-group "log"
when
        $p : Person ()
        $e : LogInEvent(person == $p)
then
        Logged ooo = new Logged();
        ooo.setPerson($p);
        insert(ooo);
end
```

```
rule "log in with premium"
        activation-group "log"
when
        $p : Person ()
        $e : LogInEvent(person == $p)
then
        Logged ooo = new Logged();
        ooo.setPerson($p);
        insert(ooo);
        System.out.println("Congrats!...");
end
```