Introduction to Jess

The Jess Language Part 1

CS3025, Knowledge-Based Systems

Lecture 03

YUTING ZHAO

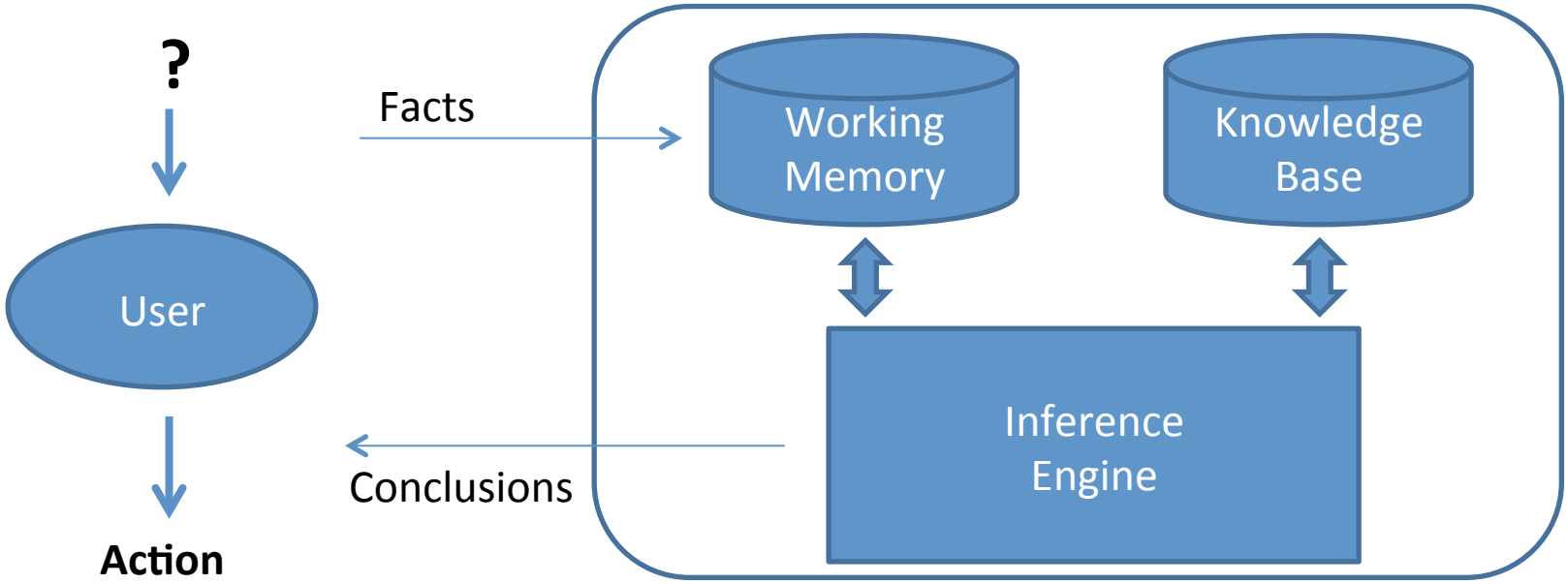
yuting.zhao@gmail.com

2017-09-19

Overview

- Jess Expert System Shell, a Forward Chaining Inference engine (backward chaining inference)
- Jess Language
  - Facts
  - Variables and their scope
  - Structured Knowledge (Jess templates)
- How to program simple expert systems

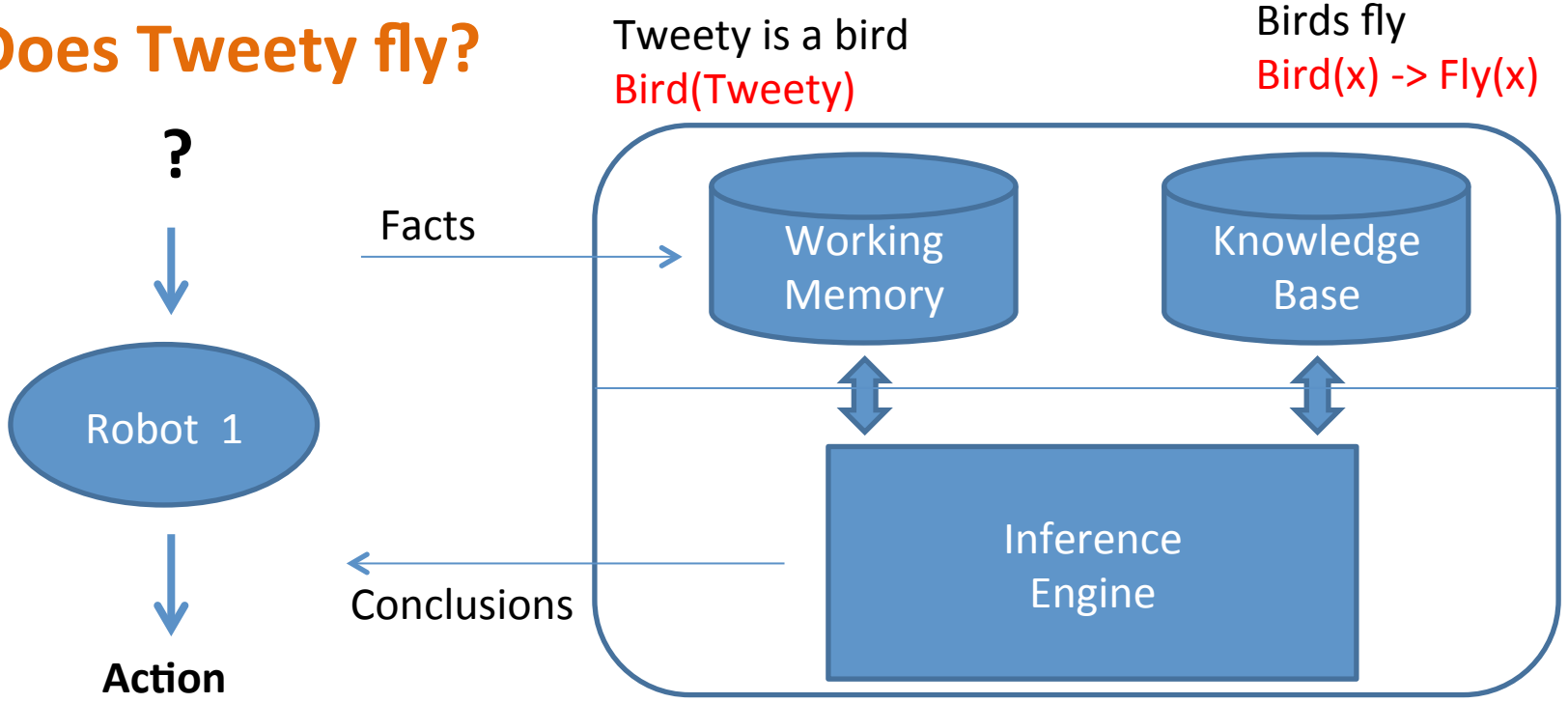
Expert System Shell



- Expert System Shells are used to implement Expert Systems
  - Separation between ***knowledge*** and ***inference engine***
  - Knowledge represented as a set of rules (the production system)
  - Inference engine is the execution mechanism that manages rule activation, execution and the facts produced

An example showing how it works

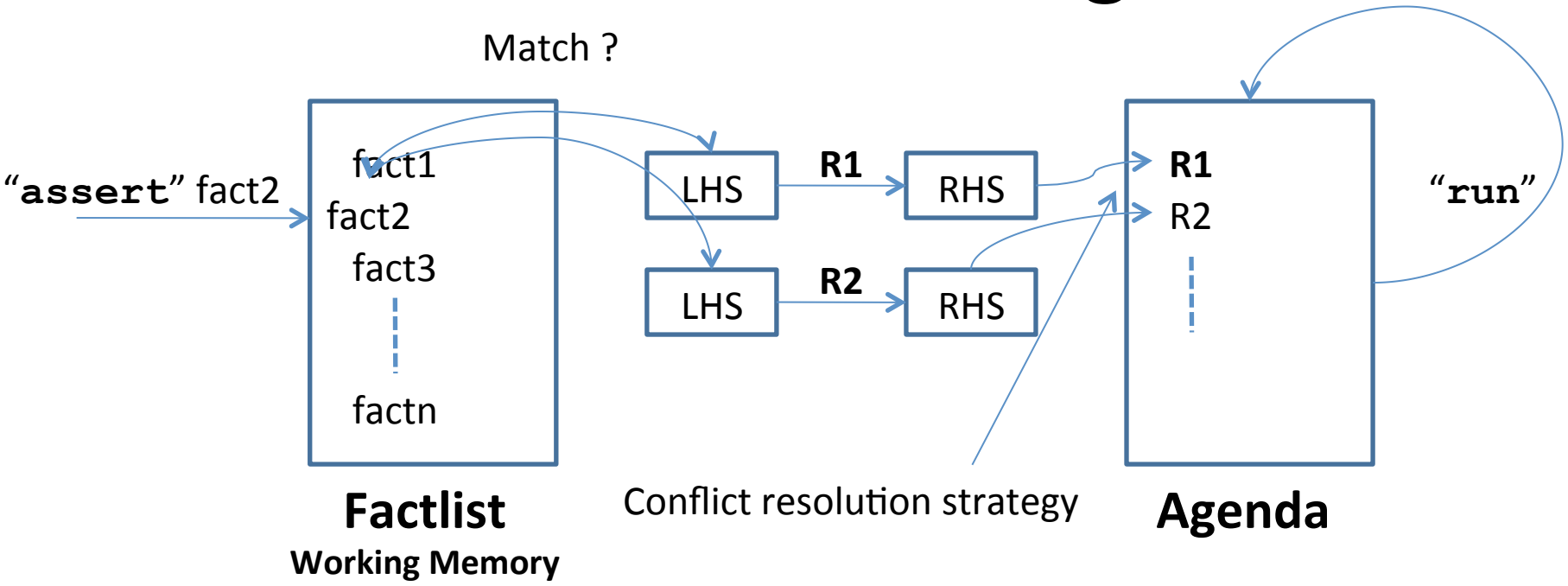
Does Tweety fly?



Yes it flies!!!

Thinking:
(1) Bird(x) matches Bird(Tweety)
(2) Bind x with Tweety
(3) Activate a rule:
Bird(Tweety) -> Fly(Tweety)
(4) Fire the rule
(5) New result: Fly(Tweety)

How does it work in general



- Two important data structures
  - **Factlist**: holds set of facts – our "Working Memory
  - **Agenda**: holds set of "activated Rules – LHS of these rules match facts on the Factlist (Working Memory)
- Rule Firing (Execution)
  - RHS of rules written to the Agenda are executed
  - Execution of rules according to sequence on Agenda

Rule Activation and Execution

- Rule Activation
  - The Agenda is populated with "activated" rules due to assertion of facts
  - Rules are deactivated and removed from the agenda, if the matching facts are removed
- Rule Execution (Firing):
  - A rule fires, if Jess executes its RHS action
  - **Sequence** of actions determined by Agenda
  - An action can be: new assertions of facts, removal / modification of facts
- Observation:
  - The content of the Agenda can be compared to a "program" that is executed (but this "program" can change during execution)

Is the sequence important? Or it does not matter?

The Jess Language

What are JESS commands?

- Programming expert systems in the Jess language
  - Uses as Lisp-like syntax: language constructs are lists (sequences of symbols enclosed by brackets)
  - Arithmetic expressions are written in a **prefix notation**:
    - E.g.: `(+ (* 2 5) (- 10 5))`
  - Jess programs use language constructs like "defrule", "assert", etc.
  - Each language construct is written as a list of lists, with the first element of this list the name of a function that takes the rest of the list as its parameters:

```
(defrule find_35_year_old_men
    (person
        (name    ?name)
        (age     35)
        (gender  male))
    =>
    (printout t "Name is " ?name crlf))
```

```
(assert (creature eats grass))
```

Ordered Facts

- Simple, so-called “ordered” facts in Jess are formulated in the following way:

  (<field> <field> <field> …)

  - Is enclosed in round brackets … “(“ “)”
  - The first field must be a symbol
  - Fields must be constants (symbols, strings, integer, float)
  - Ordering of fields is important
- For example:
  - `(creature body-covering is hair)`
  - `(creature eats grass)`
  - `(creature colour is black and white)`
- Two different ordered facts:
  - **(person Michael male 35 Diana Philip Julia)**
  - **(person Michael 35 male Diana Philip Julia)**
- Asserting such a fact into working memory:

```
Jess> (assert (person Michael male 35 Diana Philip Julia))
<fact-0>
```

“Unordered” Facts based on Templates

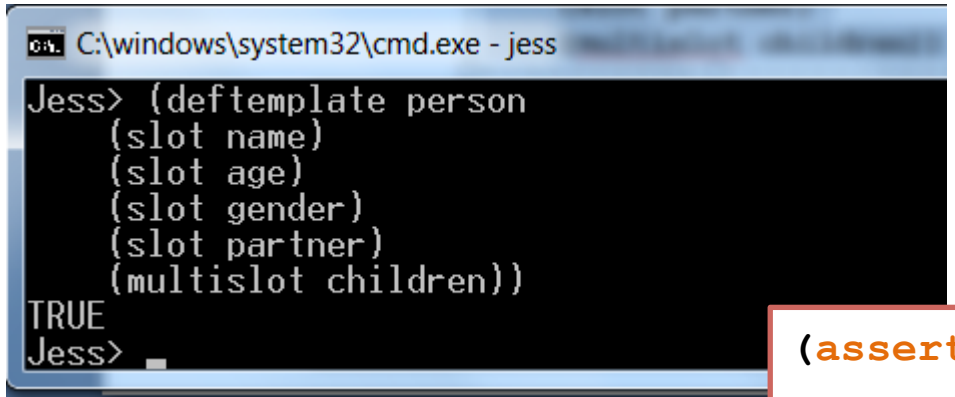
- Jess templates are used to specify structured data that can be used as facts:

```
(deftemplate person
     (slot name)
     (slot age)
     (slot gender)
     (slot partner)
     (multislot children))
```

```
(assert (person
                    (name Michael)
                    (gender male)
                    (age 35)
                    (partner Diana)
                    (children Philip
Julia)))
<fact-0>
```

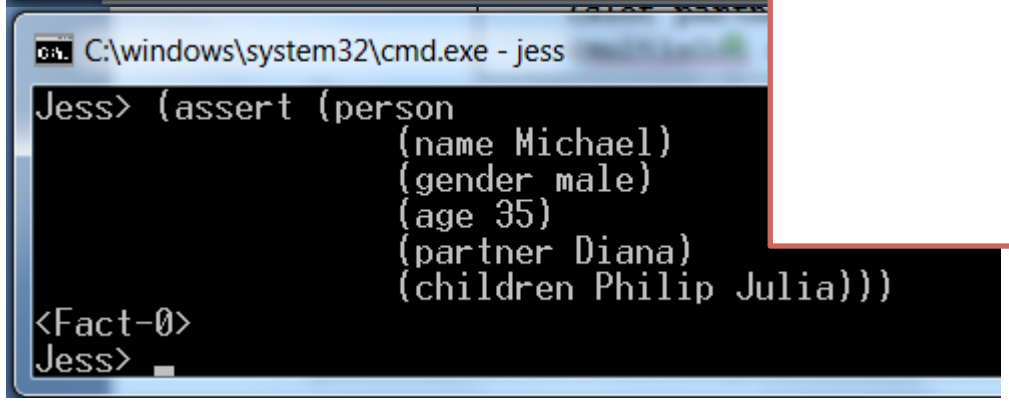
How many facts defined here??? 5? or 6? or 1?

Let’s check it now (example 2)



1 fact !!!
But ….

```
(assert (person
                    (name M2)
                    (gender male)
                    (age 35)
                    (partner D2)
                    (children P2 J2)))
```



Specifying Rules

```
(defrule bird
       (body-covering is feathers)
       (lays eggs)
       =>
       (assert (class-of bird)) )
```

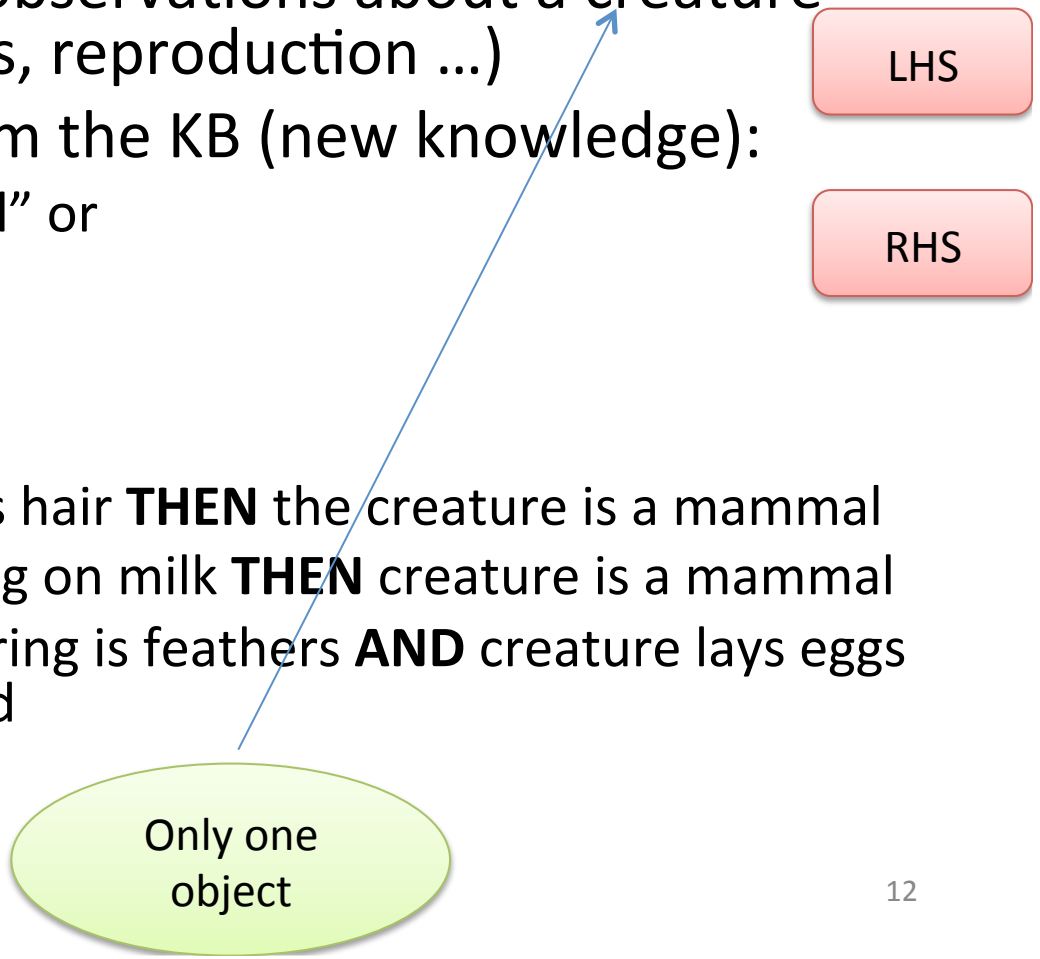
Both patterns must be in WM for the rule to fire

sequence matters?
LHS?
RHS?

- Patterns on the left-hand side:
  - All patterns must match – matching takes place in the order the patterns are written
- To match this rule (and activate it), the following facts must be in WM:
  - **(body-covering is feathers)**
  - **(lays eggs)**

Jess Forward Chaining: Creatures Example

- Building a Knowledge Base, HOW and WHY:
  - Given input (FACTS): observations about a creature (colour, hair / feathers, reproduction …)
  - Expected answers from the KB (new knowledge):
    - "creature is a mammal" or
    - "creature is a bird" or
    - "creature is a Tiger"
  - Knowledge (RULES):
    - **IF** the body-covering is hair **THEN** the creature is a mammal
    - **IF** creature feeds young on milk **THEN** creature is a mammal
    - **IF** creature body-covering is feathers **AND** creature lays eggs **THEN** creature is a bird



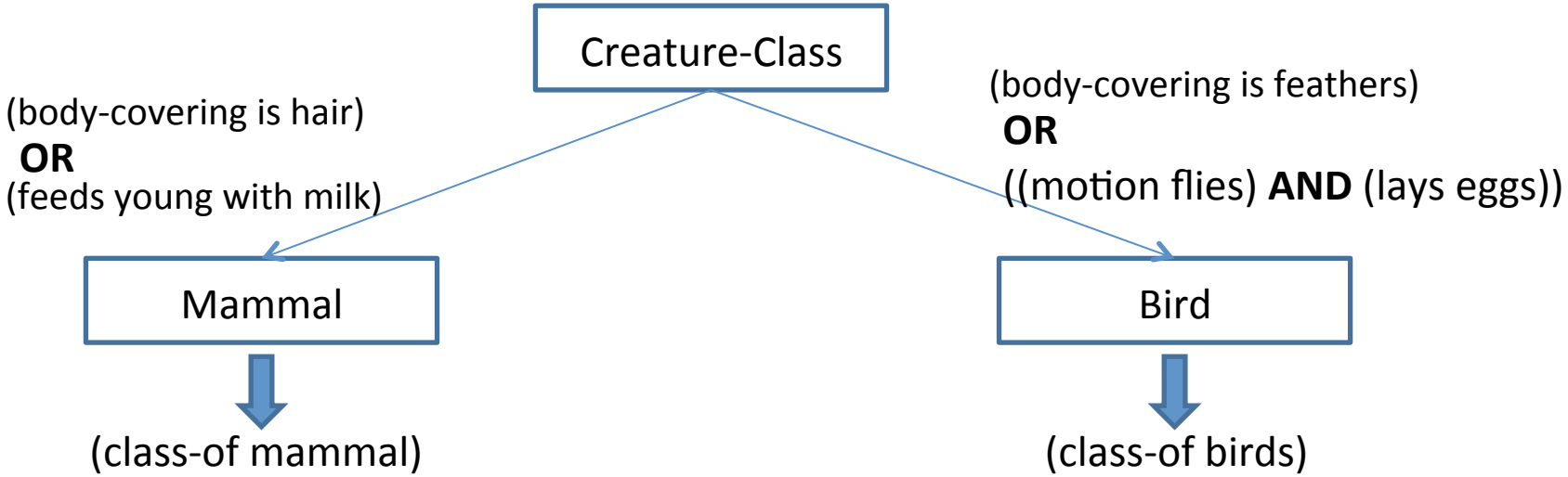
Example: Identify Creature Species

Observations	Expressed in the Jess Language
“the observed creature is covered with hair”	(body-covering is hair)
“the observed creature is covered with feathers”	(body-covering is feathers)
“the observed creature feeds the young with milk”	(feeds young with milk)
“the observed creature lays eggs”	(lays eggs)
“the observed creature eats grass”	(eats grass)
“the observed creature eats meat”	(eats meat)
“the observed creature has hoofs”	(has hoofs)
“the observed creature has clawed feet”	(has claws)
“the observed creature is black and white”	ETC. ...

- Expert System should answer: What species are we observing?

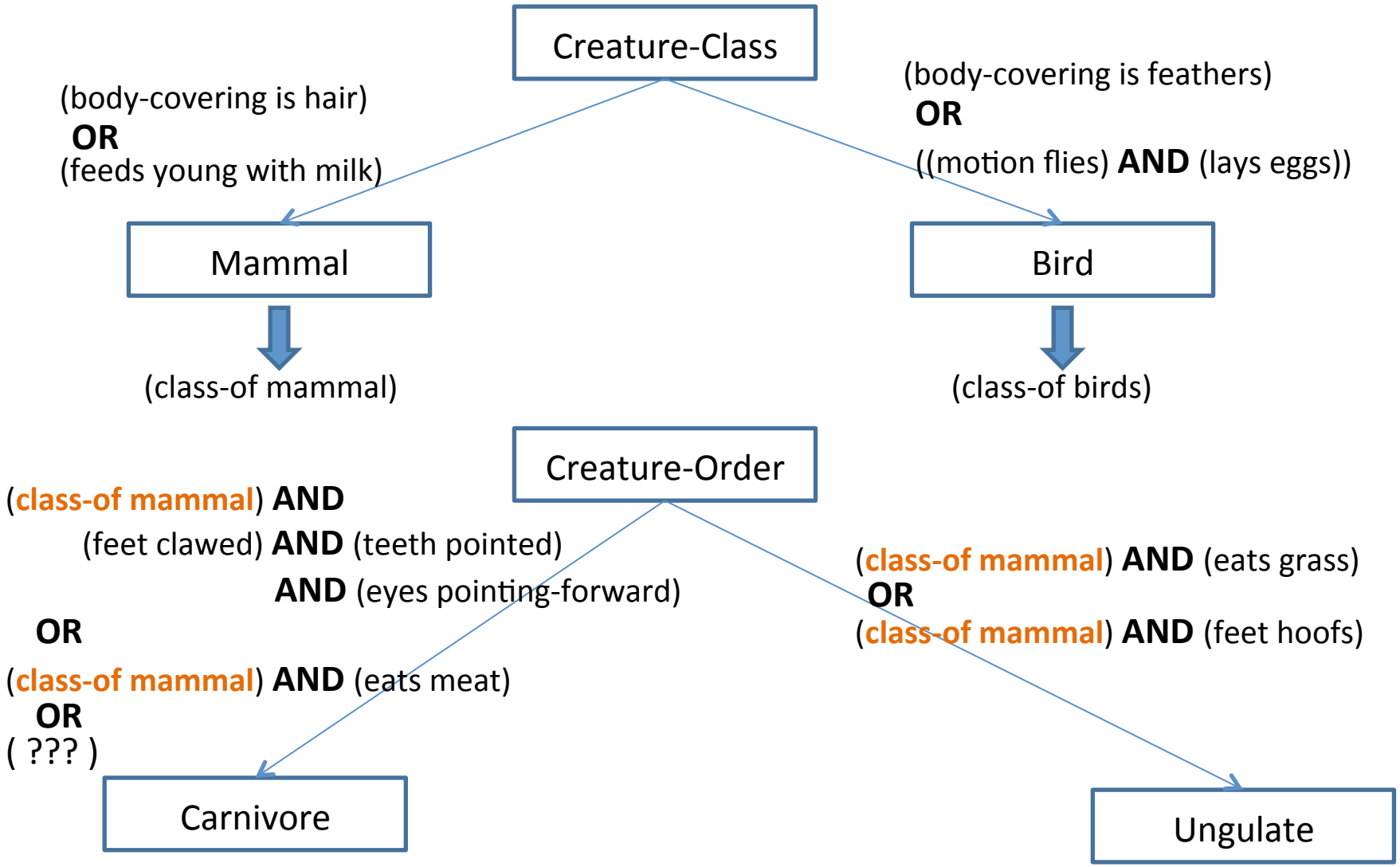
Not natural language, JESS language!!

Example: Identify Creature Species

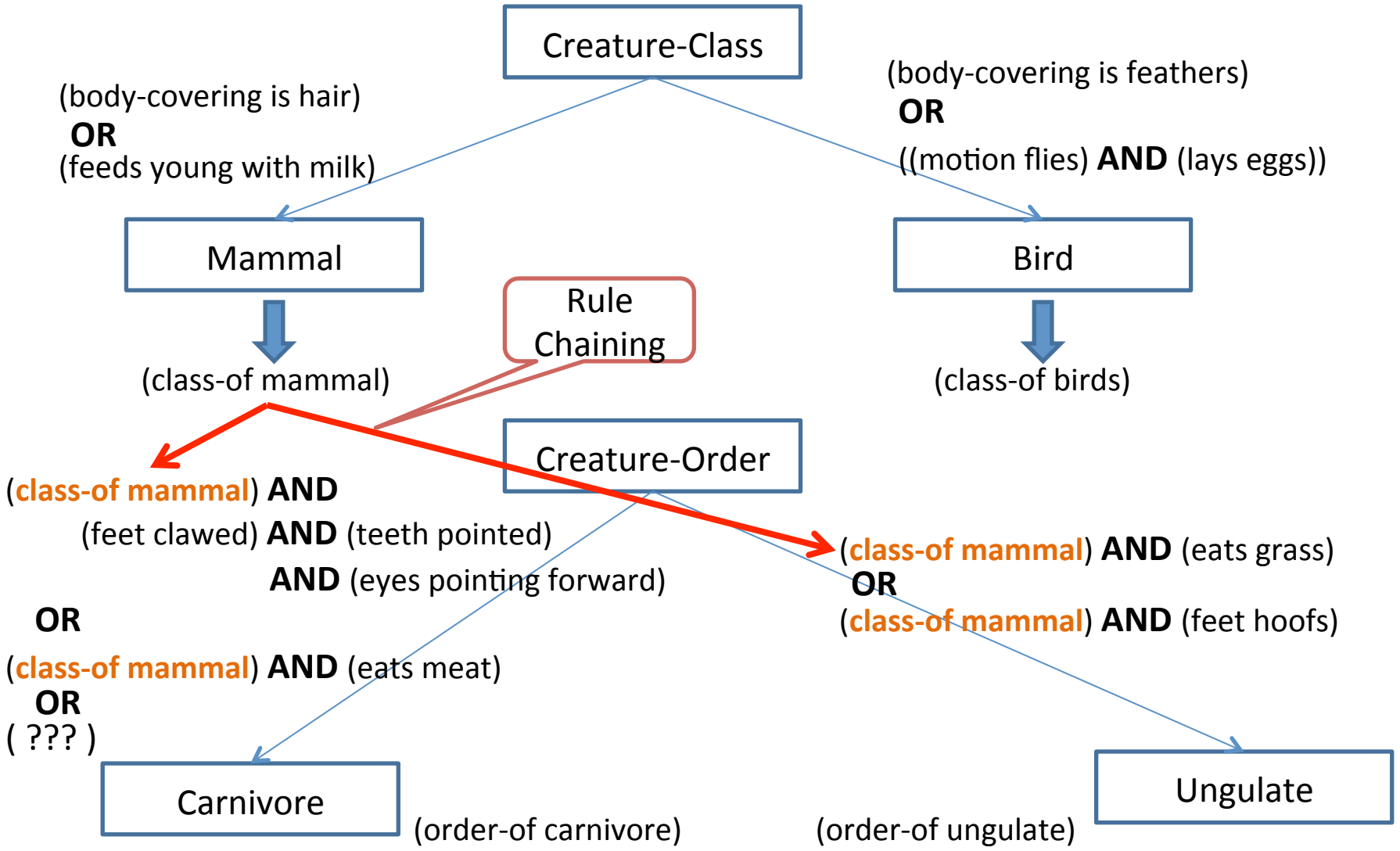


- Let's start with identifying features that classify observed creatures

Example: Identify Creature Species



Example: Identify Creature Species

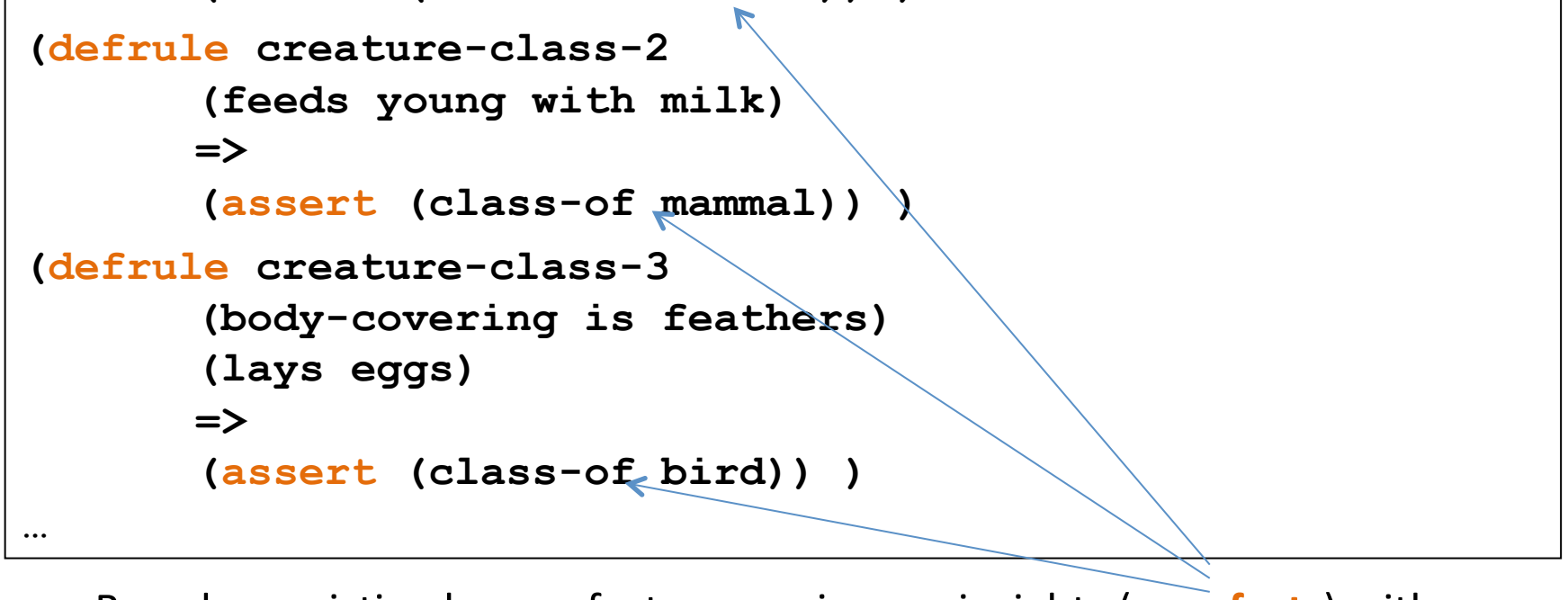


Example: Identify Creature Species

```
(defrule creature-class-1
       (body-covering is hair)
       =>
       (assert (class-of mammal)) )
(defrule creature-class-2
       (feeds young with milk)
       =>
       (assert (class-of mammal)) )
(defrule creature-class-3
       (body-covering is feathers)
       (lays eggs)
       =>
       (assert (class-of bird)) )
...
```

pre-condition

action



- Based on existing known facts, we gain new insights (**new facts**) with these rules

Example: Identify Creature Species

```
(defrule creature-order-1
       (class-of mammal)
       (feet clawed)
       (teeth pointed)
       (eyes pointing forward)
       =>
       (assert (order-of carnivore)) )
(defrule creature-order-2
       (class-of mammal)
       (eats meat)
       =>
       (assert (order-of carnivore)) )

ETC …
```

- Based on existing known facts, we gain new insights (new facts) with these rules

Example: Identify Creature Species

```
(defrule creature-species-1
        (order-of carnivore)
        (colour-of tawny)
        (marking-of black-stripes)
  =>
   (printout t "Observed species is a Tiger!" crlf)
)
ETC …
```

- Finally, after various rules have produced intermediate conclusions (added new facts to WM), we can draw a final conclusion.

Variables in Jess

- In Jess, all variables start with a **question mark**. In the following example, **`?person`** is the variable:

```
(defrule mortality
        (is-a ?person man)
        =>
        (assert (is-a ?person mortal))
```

- Jess does the following
  - It takes a pattern of the LHS of a rule:
    - E.g.: **`(is-a ?person man)`**
  - It matches this pattern against all facts in WM – it may find a match:
    - E.g.: **`(is-a Plato man)`**
  - In matching the pattern against the fact, the variable **`?person`** is bound to the value **`Plato`** – **match according to position**
  - If this rule fires then the following fact is asserted as the consequence:
    - **`(is-a Plato mortal)`**

Scope of a Variable

- In general: The scope of a variable is the area of its declaration within a program
- In Jess: A rule is the scope of a variable
  - Variables are assigned values in the antecedent (**LHS**) of the rule via pattern matching
  - These values of variables are then used in the consequent (**RHS**) of the rule
- (Compare this to Java: a variable can be declared for the whole class or only within a particular "{" ... "}" block – this is called the scope of the variable)

Variables: Universal Instantiation

- Let's modify Modus Ponens by introducing universal instantiation:

  Modus Ponens (MP):

  From: ∀x if x is-a p then x is-a q
  S is-a p
  Infer: S is-a q

- This is just the same as MP, but it shows how we can write more general rules using universally quantified variables like "x"
  - "x" is bound to the constant "S"
  - In Jess, we would write: `(bind ?x "S")`

    In Java, we would write: x = "S" ;
- Variables allow us to reason over sets of facts and draw sets of conclusions (regard each rule to be "instantiated" or "activated" for each matching fact)

Effect of Variable Binding

- Rule with more complex LHS:

```
(defrule mortality
        (is-a ?person man)
        (is-a ?person Ancient-Greek)
        =>
        (assert (speaks ?person Ancient-Greek))
```

- WM contains only one fact:
  - E.g.: `(is-a Socrates man)`
- This is not enough for a match, the rule only fires if both parts of the LHS find matching facts
  - E.g.: `(is-a Socrates Ancient-Greek)`

Multiple Activations of Rules

- If we load the following rule into Jess

```
(defrule mortality
        (is-a ?person man)
        =>
        (assert (is-a ?person mortal))
```

- If we have the following facts in our WM
  - `(is-a Socrates man)`
  - `(is-a Caesar man)`
  - `(is-a Cleopatra woman)`
- What activations of the rule are created?

Multiple Activations of Rules

```
(defrule mortality
       (is-a ?person man)
       =>
       (assert (is-a ?person mortal))
```

```
(is-a Socrates man)
(is-a Caesar man)
(is-a Cleopatra woman))
```

Working Memory

- The LHS of our rule "mortality" matches now twice against the WM – we get two activations of rule "mortality":
  - Activation 1: **?person** is bound to value "**Socrates**"
    - `(is-a Socrates man)`
  - Activation 2: **?person** is bound to value "**Caesar**"
    - `(is-a Caesar man)`
- If we run this Jess program, we therefore get two inferences from the same rule:
  - `(is-a Socrates mortal)`
  - `(is-a Caesar mortal)`

Activation of Multiple Rules

- Suppose we have two rules:

```
(defrule mortality
        (is-a ?person man)
        =>
        (assert (is-a ?person mortal)) )

(defrule hairy
        (is-a ?human man)
        =>
        (assert (can-grow ?human beard)) )
```

- The WM contains only one fact
  - **(is-a Socrates man)**
- What activations are created?

Activation of Multiple Rules

- With one fact, we get two activations – the LHS of both rules match the fact in WM:
  - Activation 1 has **`?person`** bound to value "**`Socrates`**"
  - Activation 2 has **`?human`** bound to value "**`Socrates`**"
- If we run this Jess program, we therefore get two inferences from two different rules:
  - `(is-a Socrates mortal)`
  - `(can-grow Socrates beard)`

Creatures again

- We can now modify the creatures knowledge base by introducing variables:

```
(defrule mammal-1
        (body-covering ?creature hair)
        =>
        (assert (class-of ?creature mammal) ))

(defrule feeding-type-1
        (class-of ?creature mammal)
        (eats ?creature meat)
        =>
        (assert (order-of ?creature carnivore) ))
```

Creatures again

Working Memory

```
(deffacts more-creatures
          (body-covering Shere-Khan hair)
          (eats Shere-Khan meat)
          (colour-of Shere-Khan tawny)
          (marking-of Shere-Khan black-stripes)
          (body-covering Pengi feathers)
          (motion Pengi swims)
          (motion Pengi waddles)
          (colour-of Pengi black-and-white) )
```

"deffacts" A new command for facts.

Question: how many ways to put facts in JESS?

- What is **?creature** at the time of rule activation:

```
(defrule mammal-1
          (body-covering ?creature hair)
          =>
          (assert (class-of ?creature mammal) ))

(defrule feeding-type-1
          (class-of ?creature mammal)
          (eats ?creature meat)
          =>
          (assert (order-of ?creature carnivore) ))
```

Jess Templates

- Jess provides templates for representing structured knowledge or facts
- Structured facts are also called "un-ordered facts" in Jess – the order of definition of slots is not relevant for matching elements in the Working Memory
- Templates are similar to Java class specifications
- Designing templates is essential for modeling knowledge bases
  - What things are there in the domain you want to model?
  - What attributes do they have?
    - Color, age, relationships to other things, etc.

```
(deftemplate person
     (slot name)
     (slot age)
     (slot gender)
     (slot partner)
     (multislot children))
```

The same in Java:

```
public class person {
    public String name ;
    public int age ;
    public char gender ;
    public String partner ;
    public String [10] children ;
}
```

deftemplate - Qualifiers

```
(deftemplate person "the person template"
        (slot name
                (type STRING))
        (slot gender
                (allowed-values male female))
        (slot age
                (type INTEGER)
                (default 35))
        (slot partner (type SYMBOL))
        (multislot children) )
```

Value can only be a string

Value can be male or female

Value can only be an integer

If no value set, defaults to 35

Value can only be a symbol

Templates

Declaration

```
(deftemplate person
     (slot name)
     (slot age)
     (slot gender)
     (slot partner)
     (multislot children))
```

Are order of slots important?

Assertion

```
Jess> (assert (person
                    (name Michael)
                    (gender male)
                    (age 35)
                    (partner Diana)
                    (children Philip Julia))
<fact-0>
```

Pattern Matching in Rules

```
(defrule find_35_year_old_men
    (person
        (name    ?name)
        (age     35)
        (gender  male))
    =>
    (printout t "Name is " ?name crlf))
```

- Match slots just like ordered facts
  - Specify (slotname, value or variable) – pairs to test what values the slots of a structured fact hold in working memory
- Use of variables as before
- Order of slots is not important, as each slot has a name
- For a match, not all slots have to be specified, a match will occur according to the slots used in the antecedent of a rule
- This rule will match with an occurrence of a structured fact in WM

How does match and bind?

name	gender	age	partner	children	children
Michael	**male**	35	**Diana**	**Philip**	**Julia**
M2	**male**	35	D2	P2	J2

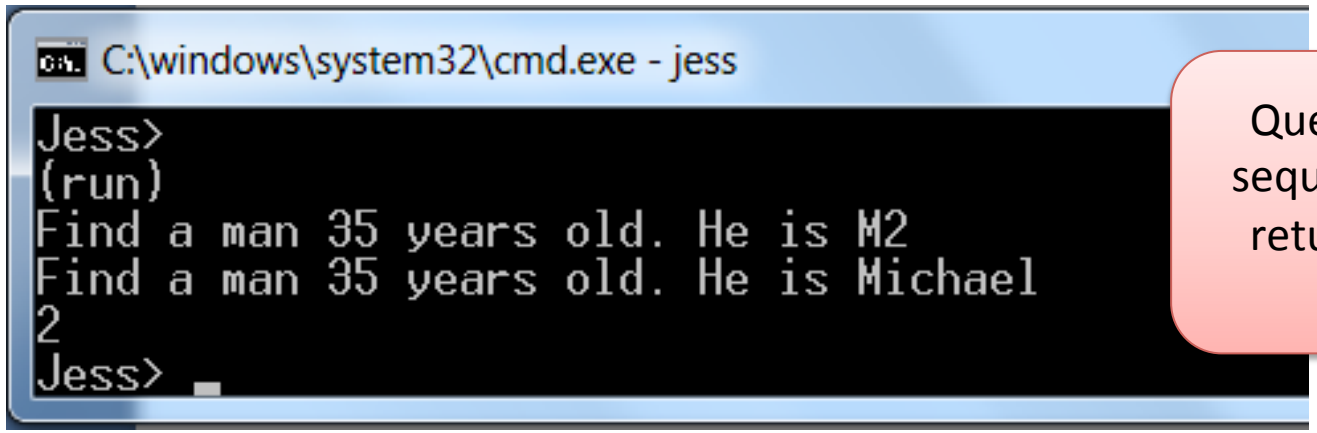
```
(defrule find_35_year_old_men
    (person
        (name    ?name)
        (age     35)
        (gender  male))
    =>
    (printout t " Find a man 35 years old. He is " ?name crlf))
```

Example 2 (Cont.)

```
(defrule find_35_year_old_men
     (person
          (name     ?name)
          (age      35)
          (gender   male))
     =>
     (printout t "Find a man 35 years old. He is " ?name crlf))
```

- Add another fact and then
- execute it (run)

Looking at the LHS, it matches HALF fact !!!



Question: look at the sequence, why is "M2" returned earlier than "Michael"?

summary

- What are JESS commands?
- What are LHS and RHS?
- When building knowledge systems, what 3 elements we should consider carefully?
- Variables, and the scope of a variable
- Sequence, important or not?

THANKS!