Revision of Jess (2)

CS3025, Knowledge-Based Systems
Lecture 21

Yuting Zhao
Yuting.zhao@gmail.com

2017-11-28

What we have learnt

- Jess Efficiency and Pete network
- backward chaining
- Reasoning with uncertainty
- Learning and decision tree
- Access control & implementation with Jess

Jess Efficiency

The Jess Language Part 4

CS3025, Knowledge-Based Systems

Lecture 13

Yuting Zhao

Yuting.zhao@gmail.com

2017-10-31

Relationships between Patterns

- **Intra**-pattern Relationship
- constrains
- **inter**-pattern relationship

```
(deftemplate myfirst
                (slot a)(slot b)(slot c))
(deftemplate mysecond
                (slot d)(slot e))
(deftemplate mythird
                (slot f))
```

1 Intra-pattern Relationship:

Fact "myfirst": Is slot a == slot b?

2 Constraints

Fact "myfirst": Is slot c == "somevalue"?

?x

```
(defrule rule-1
    (myfirst    (a ?x)(b ?x)(c somevalue)
    (mysecond   (d ?x))
    =>
    (printout t "matched first "
                "and second" crlf)
)
```

?x

Inter-pattern Relationships

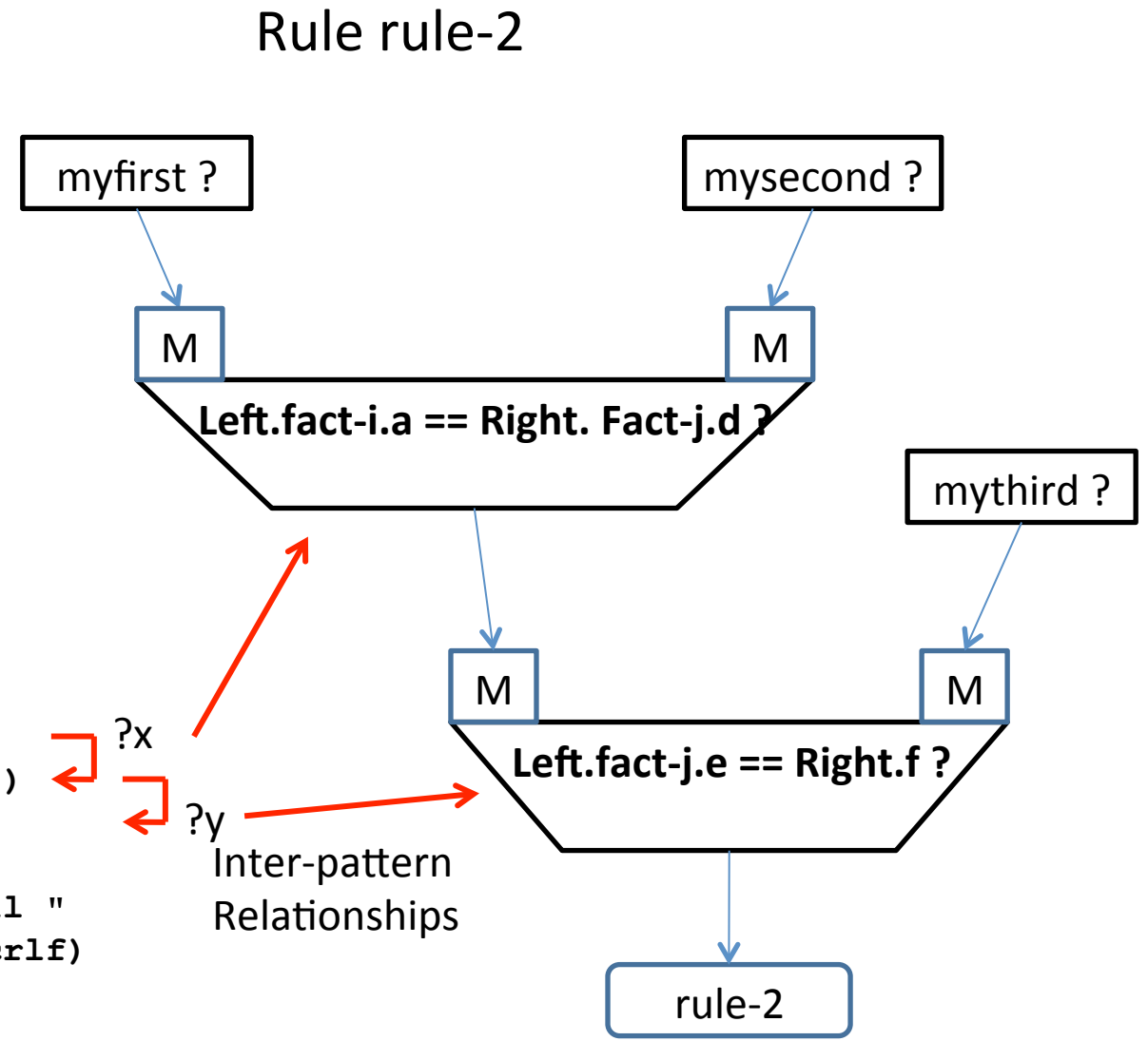
3

Compare fact "myfirst" with fact "mysecond":
Is slot **a** of "myfirst" == slot **d** of "mysecond" ?

Compile Rule into a Rete Network

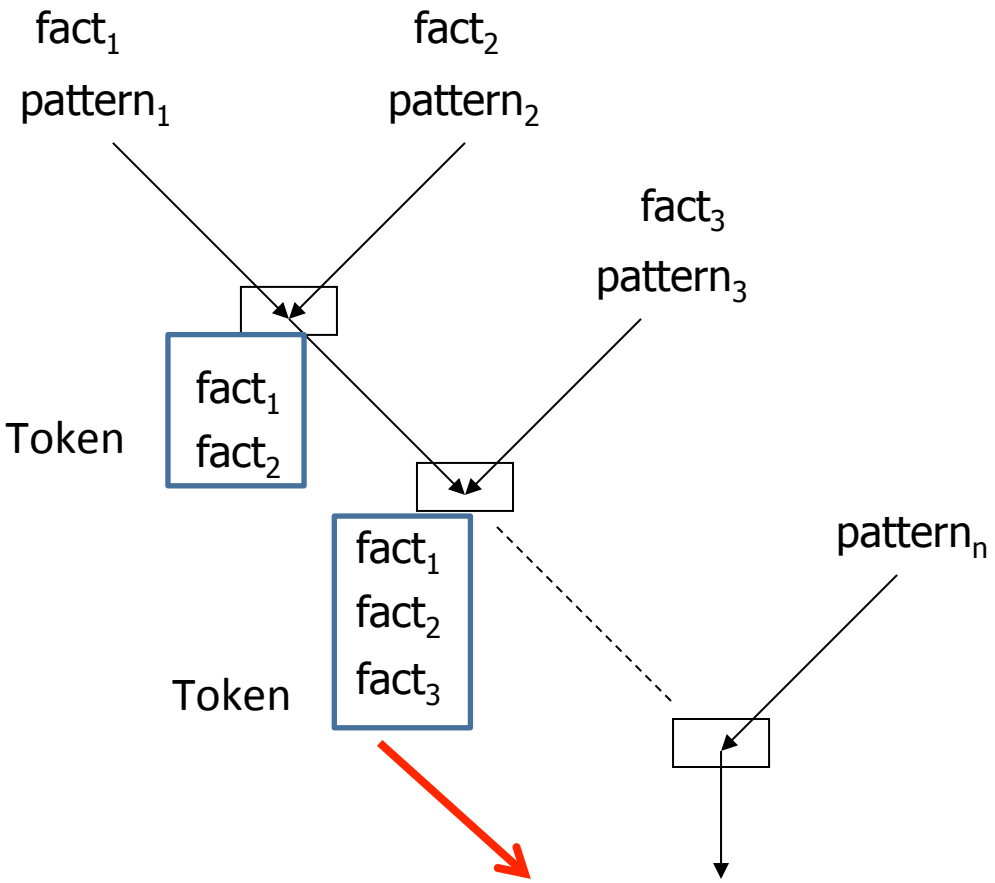
```
(deftemplate myfirst
                (slot a)
                (slot b)
                (slot c)
(deftemplate mysecond
                (slot d)
                (slot e))
(deftemplate mythird
                (slot f))
```

```
(defrule rule-2
     (myfirst    (a ?x))
     (mysecond   (d ?x)(e ?y))
     (mythird    (f ?y))
     =>
     (printout t "matched all "
                 "of them" crlf)
)
```



Rete Network

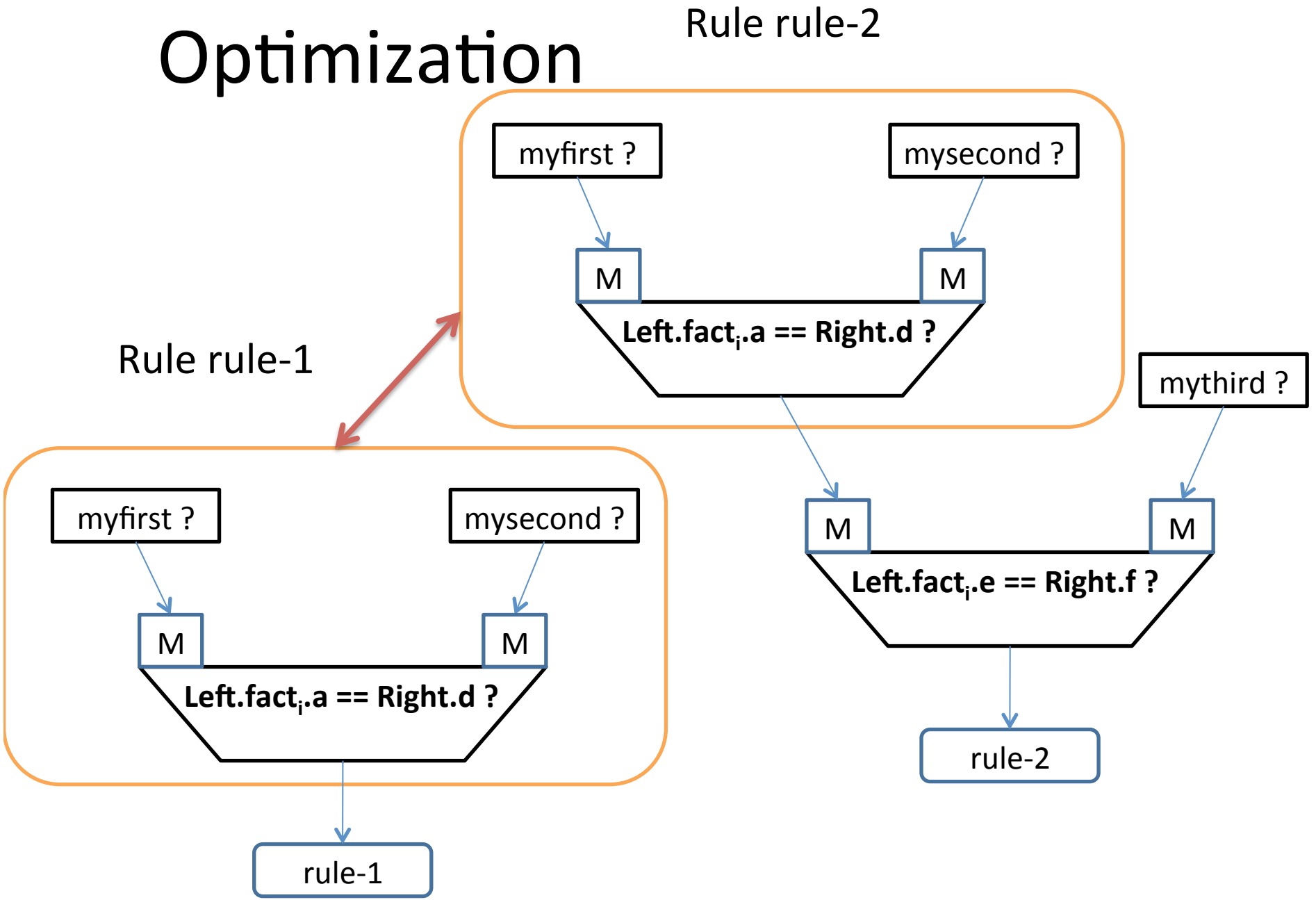
Adding new Fact, **execute Tests** in Join Node



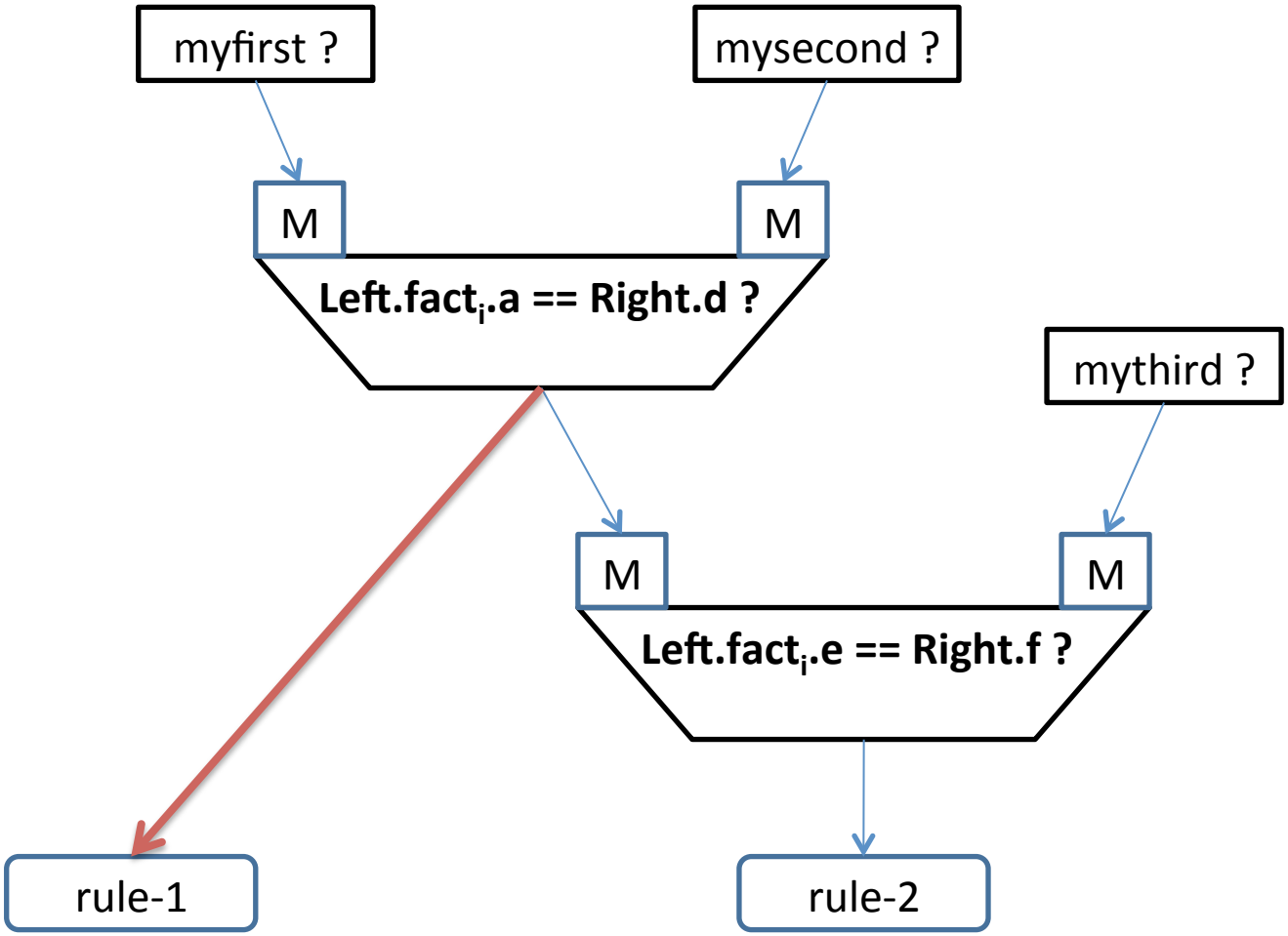
- Adding a new fact for pattern 3 will lead to the execution of tests in the join node

Question:
if pattern-3 is not matched, then what?

Optimization



Optimization



Jess Backward Chaining

The Jess Language Part 5
CS3025, Knowledge-Based Systems
Lecture 16

Yuting Zhao
Yuting.zhao@gmail.com

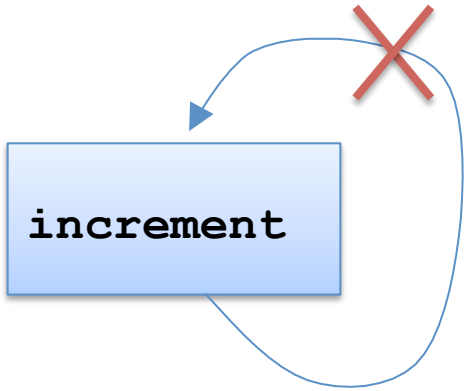
2017-11-09

Refraction
Avoid Re-activation of a Rule

- **Refraction** is the act of preventing a rule from re-activating itself

- Declaration in rule:

```
(defrule increment
     (declare (no-loop TRUE))
     ?f <- (person
              (name   ?name)
              (salary ?salary))
     =>
     (modify ?f (salary (+ ?salary 10)))
)
```



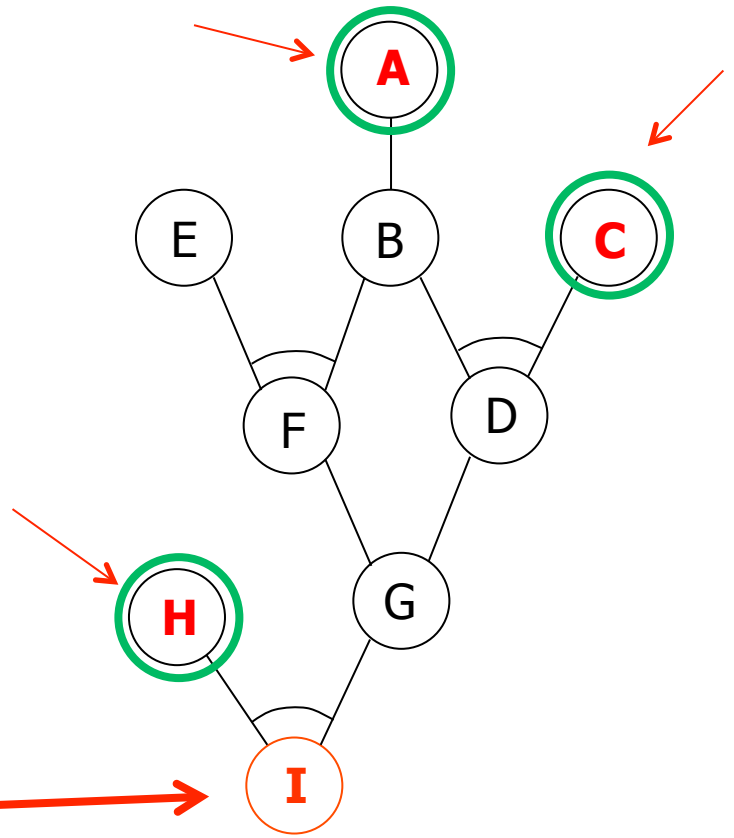
Backward Chaining

- Given a particular **hypothesis** (a fact we regard as true), which we also call a "goal"
  - We want to prove that the hypothesis is true
- For each rule whose **consequent** matches the current hypothesis:
  - Find support (matching facts) for the rule's **antecedents**
    - Match them to known facts (in WM)
    - Chain backwards through other rules that create hypotheses that, if proven, will support antecedents of your current rule
  - If all the of the current rule's antecedents are supported (match facts or supported hypotheses), then we can conclude that its consequent (and our original hypothesis) is true
- Classical language based on backward chaining: Prolog

- R1: If A then B
- R2: If B and C then D
- R3: If B and E then F
- R4: If F then G
- R5: If D then G
- R6: If G and H then I

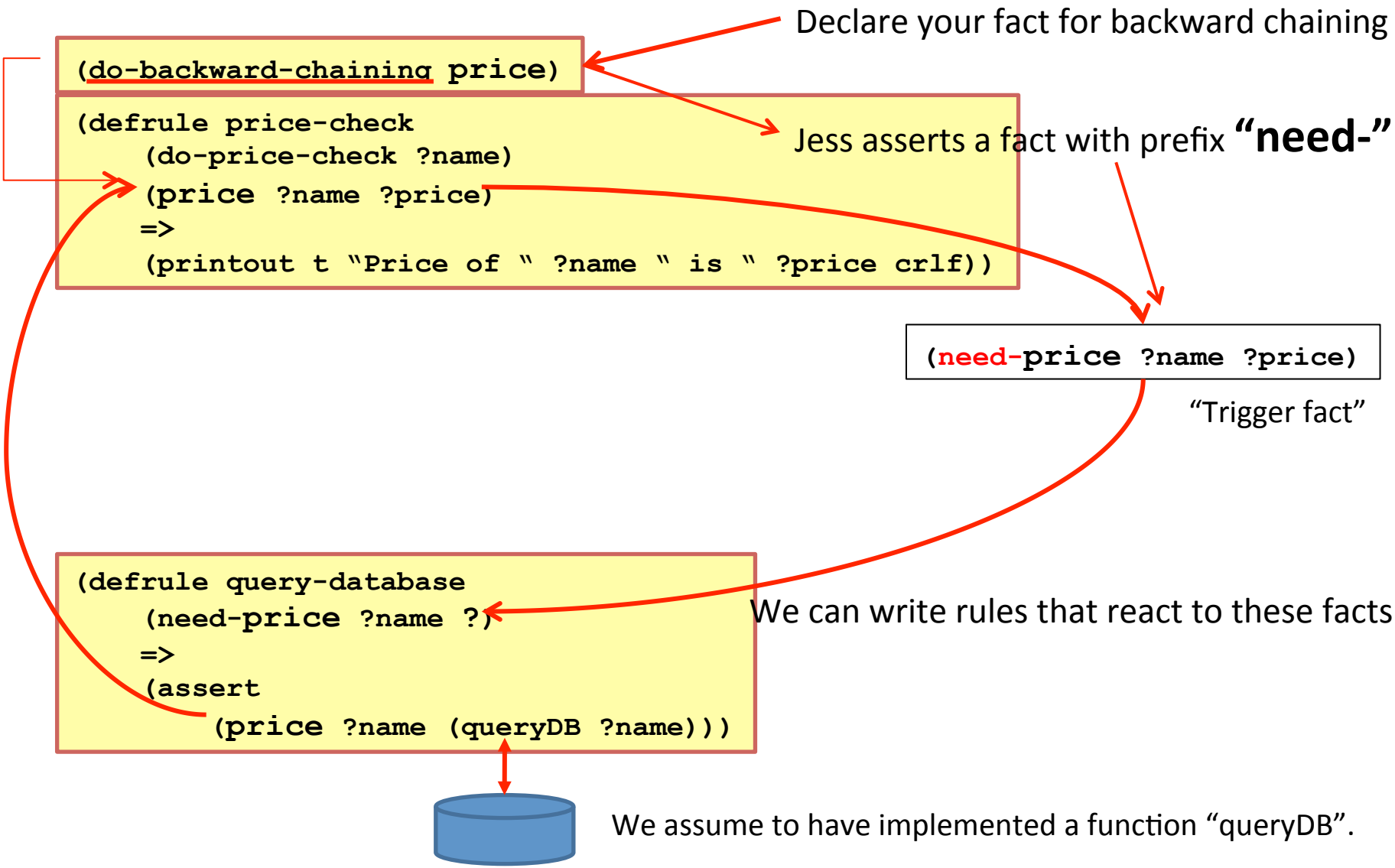
In WM: A,C and H

Our Goal: I



Question:
Can we prove I ?

Example


(do-backward-chaining price)
Declare your fact for backward chaining
(defrule price-check
(do-price-check ?name)
(price ?name ?price)
=>
(printout t "Price of " ?name " is " ?price crlf))
Jess asserts a fact with prefix "need-"
(need-price ?name ?price)
"Trigger fact"
(defrule query-database
(need-price ?name ?)
=>
(assert
(price ?name (queryDB ?name)))
We can write rules that react to these facts
We assume to have implemented a function "queryDB".

Reasoning with Uncertainty

The Jess Language Part 6
CS3025, Knowledge-Based Systems
Lecture 17

Yuting Zhao
Yuting.zhao@gmail.com

2017-11-14

MYCIN

- **Classic expert system** for the diagnosis and treatment of blood infections
  - Developed at Stanford University Shortliffe, Buchanan *et al.* from 1972
  - Important because the approach used (representation of uncertainty) has been widely copied.
- Uses so-called "**certainty factors**" to represent uncertainty

- Knowledge: facts, rules
- Value: CF [-1,1]
- Operation:
  - LHS,
  - LHS->RHS,
  - RHS

Combining Evidence

- Combining Evidence for a conclusion is done in three stages
  - Antecedent combination (**LHS**)
    - Find the minimum confidence factor of facts matched at the LHS of a rule
  - Antecedent to consequent propagation (**LHS -> RHS**)
    - Rules have their own confidence factor, combine this with the confidence factor of the antecedent
  - Multiple consequent combination (**RHS**)
    - If two rules produce the same conclusion (maybe with different confidence factors), both conclusions (their confidence factors) have to be taken into account to derive a combined confidence factor

Antecedent Combination (LHS)

- Procedure: "If there are multiple antecedents of a rule, take the minimum"
  - If the rule matches a set of facts with its LHS (each fact with a CF) in a particular activation, choose the minimum CF
  - The **minimum** CF is the CF of the antecedent (LHS) of the rule

```
(deftemplate myfact (slot name)(slot cf))

(assert (myfact (name A)(cf 0.4)))
(assert (myfact (name B)(cf 0.9)))

(defrule R1
     (myfact (name A)(cf ?x1))
     (myfact (name B)(cf ?x2))
    =>
    . . .
)
```

For example: rule R1 matches two facts:
- Fact A has a CF = 0.4
- Fact B has a CF = 0.9

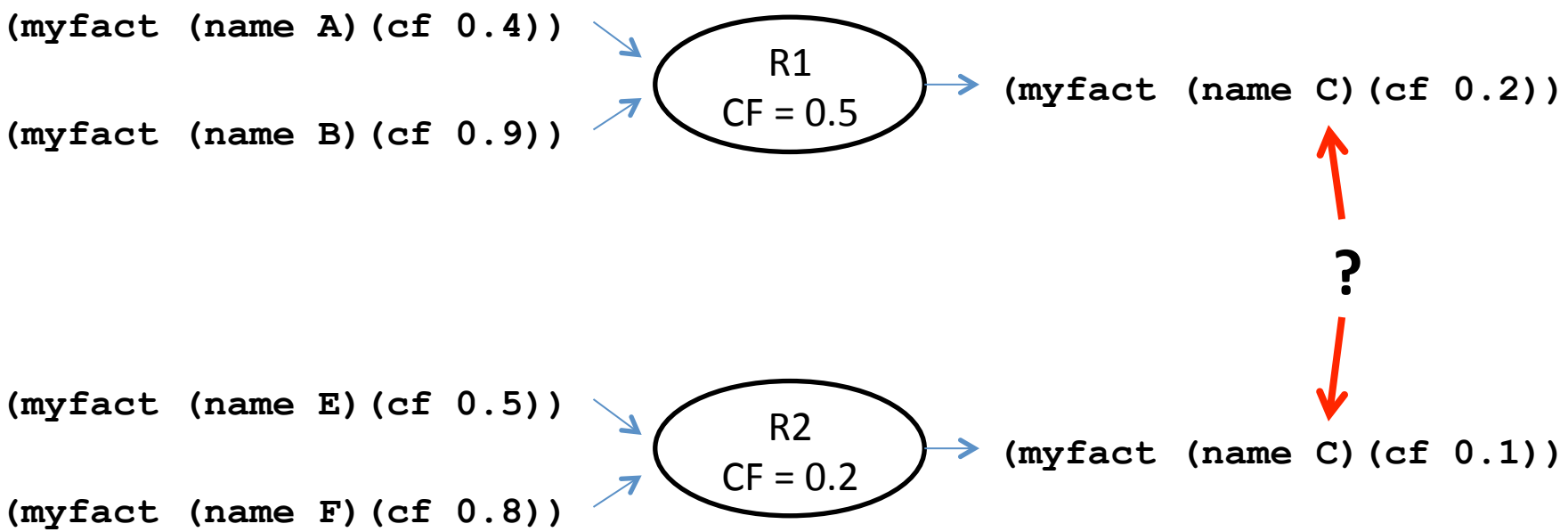
Therefore: the CF for the LHS of R1 is 0.4

Antecedent to Consequent Propagation (LHS -> RHS)

- Each rule also has a confidence factor
- In order to calculate the CF of the consequent of a rule:
  - Once the minimum CF of the LHS of a rule is determined, the consequence is:
    - **Minimum CF of LHS x Rule CF = Consequent CF**
- Example
  - The LHS CF = 0.4
  - The rule CF = 0.5
  - Therefore: the consequent CF = 0.4 **X** 0.5 = 0.2

Multiple Consequent Combination (RHS)

- This calculation has to be applied, if there are two rules that produce the same consequent



Multiple Consequent Combination

(1) Both Consequents are Positive

- If there exist two consequent evaluations, then the combined confidence factor CF of the consequent is determined in the following way:
  - Given two rules R1, R2
    - R1 produces a consequent with a confidence factor CF_1
    - R2 produces the same consequent with a confidence factor CF_2
  - If both CF_1 and CF_2 are **positive**, then use the following formula to calculate the final CF of the consequent:

$$CF = CF_1 + CF_2 \times (1 - CF_1)$$
$$= CF_1 + CF_2 - (CF_1 \times CF_2)$$

Multiple Consequent Combination

(2) Both Consequents are Negative

- If both consequent evaluations are negative:
  - Given two rules R1, R2
    - R1 produces a consequent with a negative confidence factor CF_1
    - R2 produces the same consequent with a negative confidence factor CF_2
  - Use the following formula to calculate the final CF of the consequent:

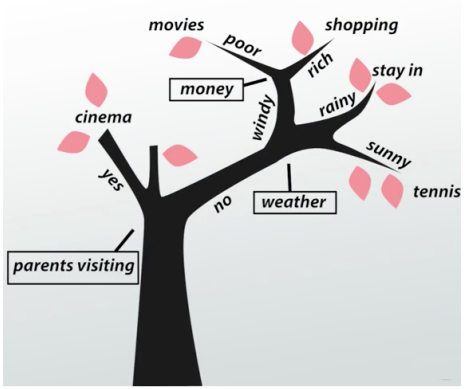
$$CF = CF_1 + CF_2 \times (1 + CF_1)$$
$$= CF_1 + CF_2 + (CF_1 \times CF_2)$$

Multiple Consequent Combination

(3) Positive and Negative CF

- If one consequent evaluation is positive and one is negative, then use the following formula to calculate the final CF of the consequent:

$$\frac{CF_1 + CF_2}{1 - \min(|CF_1|, |CF_2|)}$$



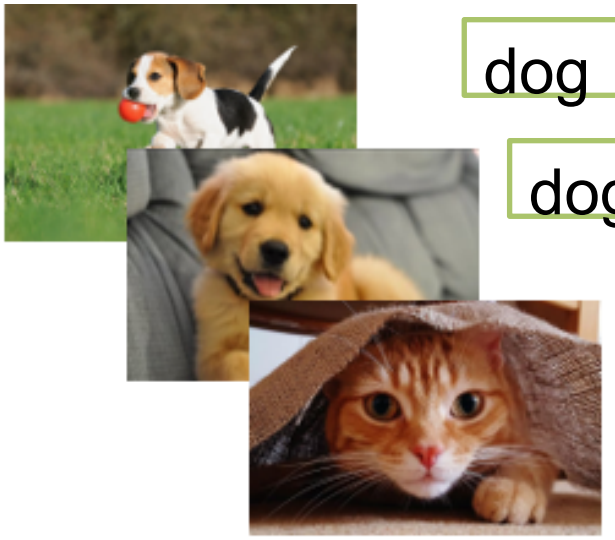
Decision Tree and Concept Learning

CS3025, Knowledge-Based Systems
Lecture 18

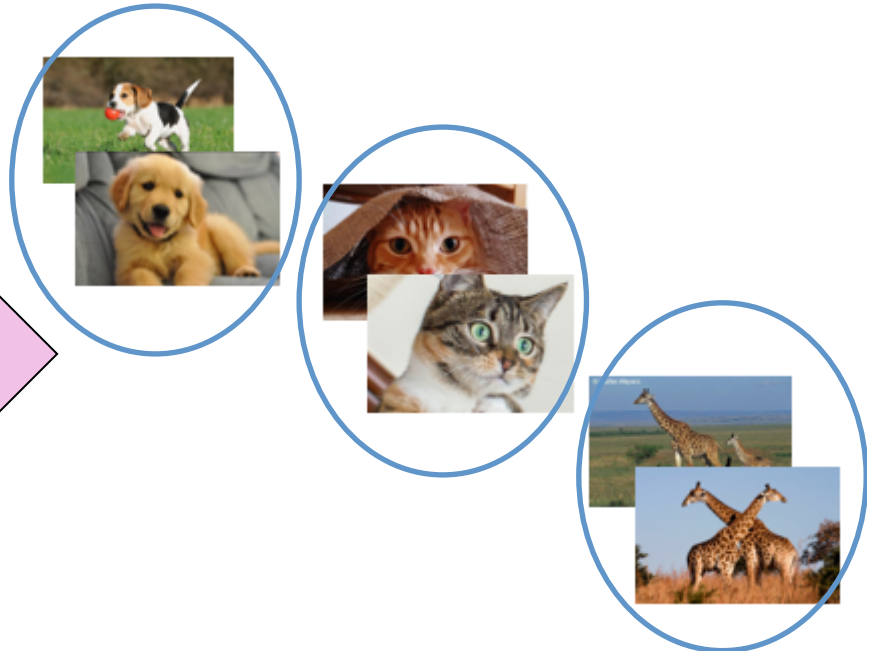
Yuting Zhao
Yuting.zhao@gmail.com

2017-11-16

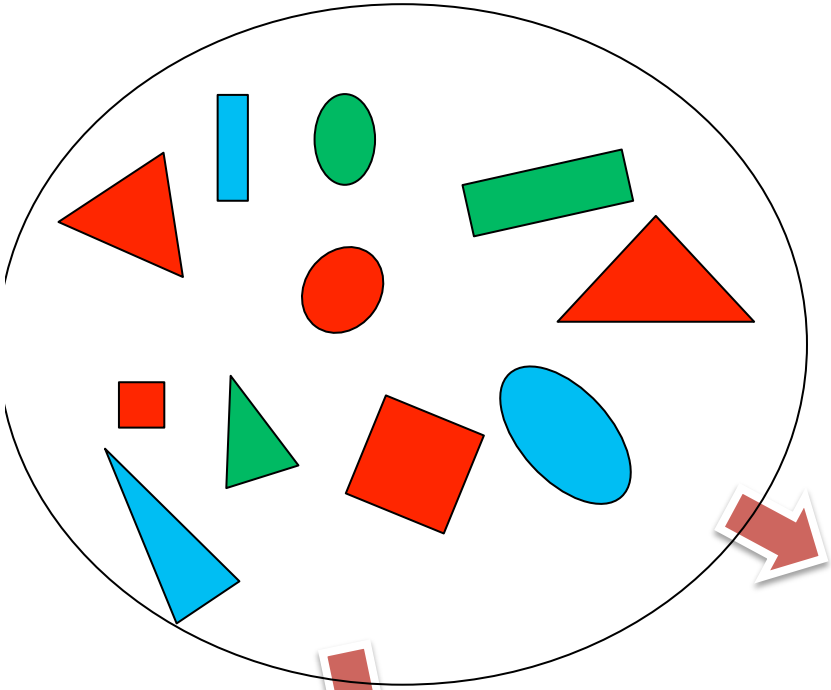
Supervised learning



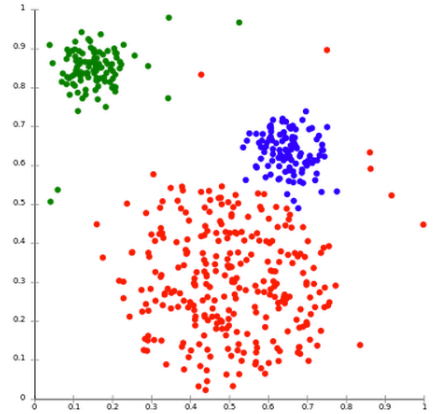
unsupervised learning

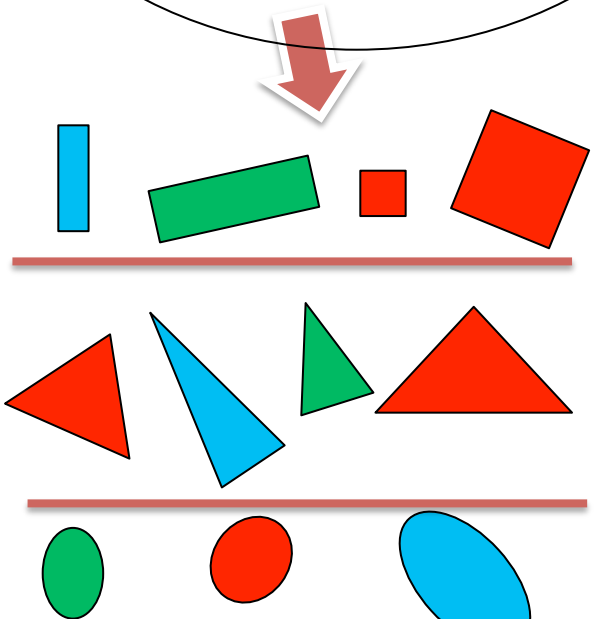


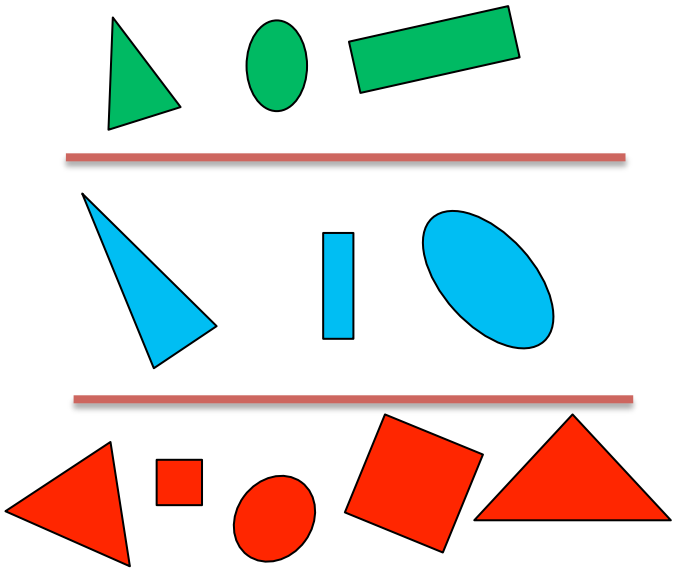
Unsupervised Learning



- Cluster analysis or clustering: putting “similar” together.
- How to define similarity



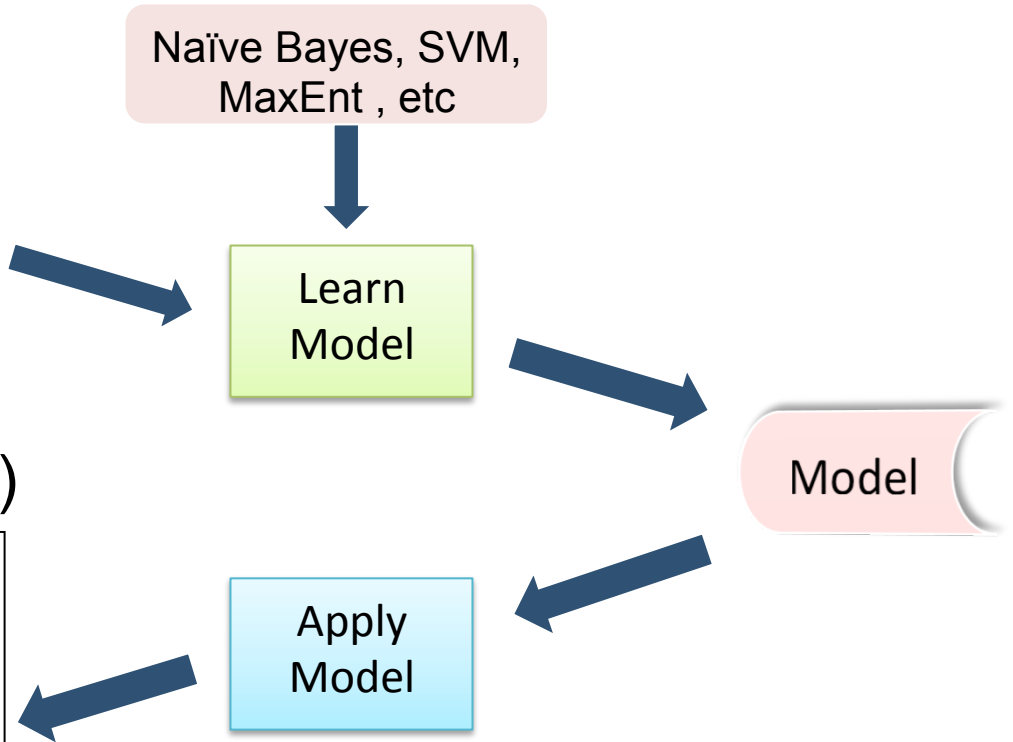




Supervised Classifier

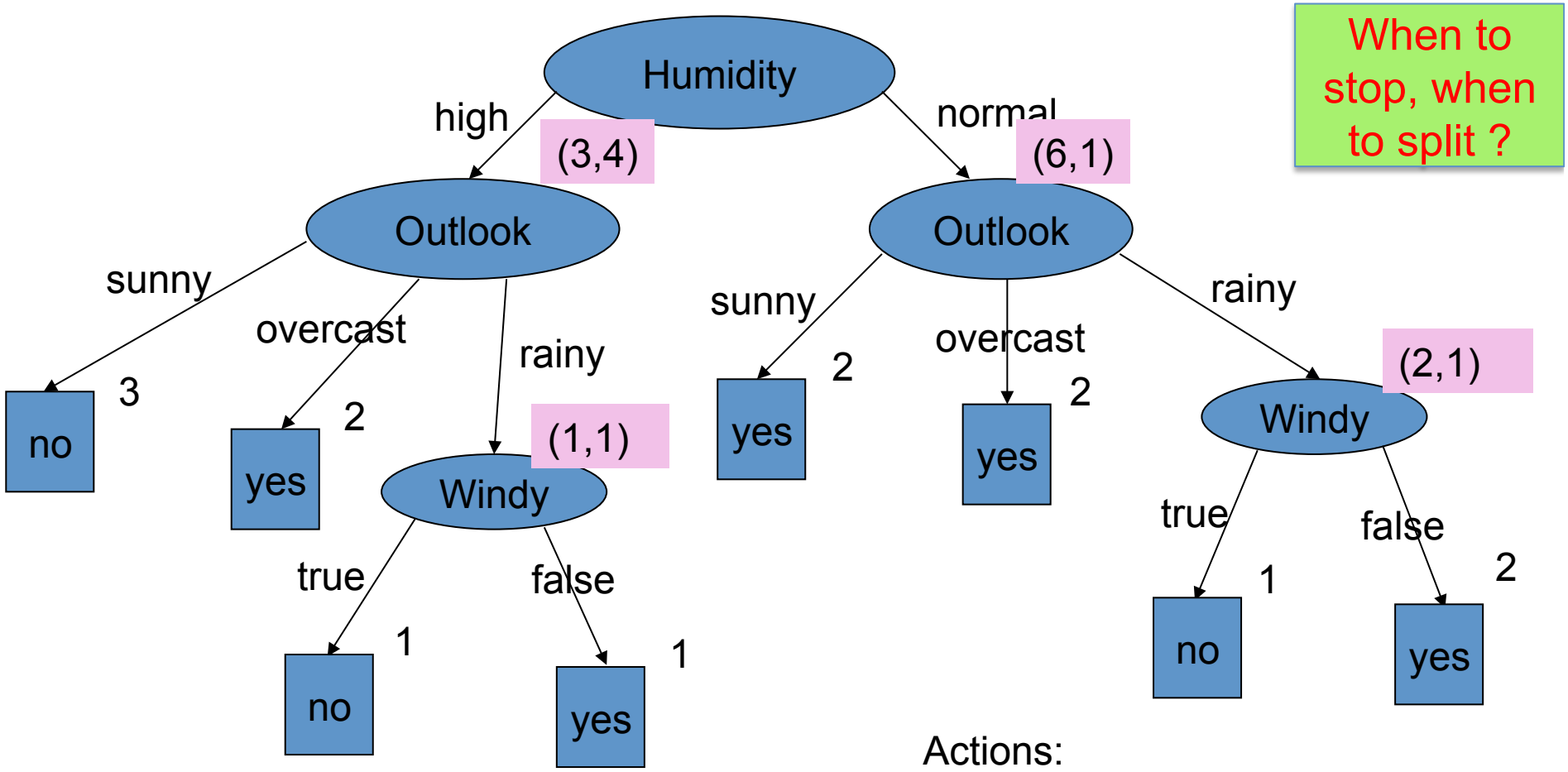
Training set (corpus)

Test set



- Rely on syntactic or co-occurrence patterns in large text corpora

Example Decision Tree 1



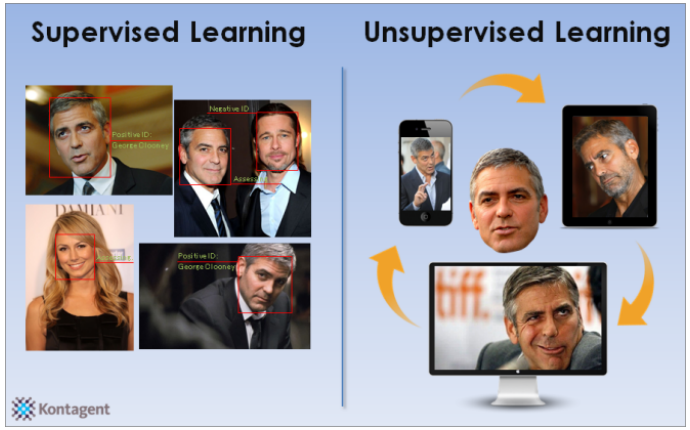
Observations:
Outlook and windy repeated in this tree
Windy tested only when outlook is rainy

Actions:

Start the tree with Outlook

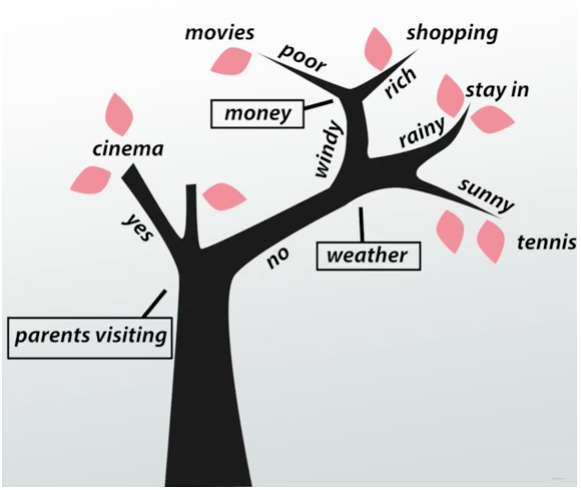
Test windy when outlook is rainy

What you should know



- Two different types of learning
  - Supervised learning (inductive learning)
  - Unsupervised learning

- Decision Tree
  - supervised learning algorithm
  - Attribute selection for splitting



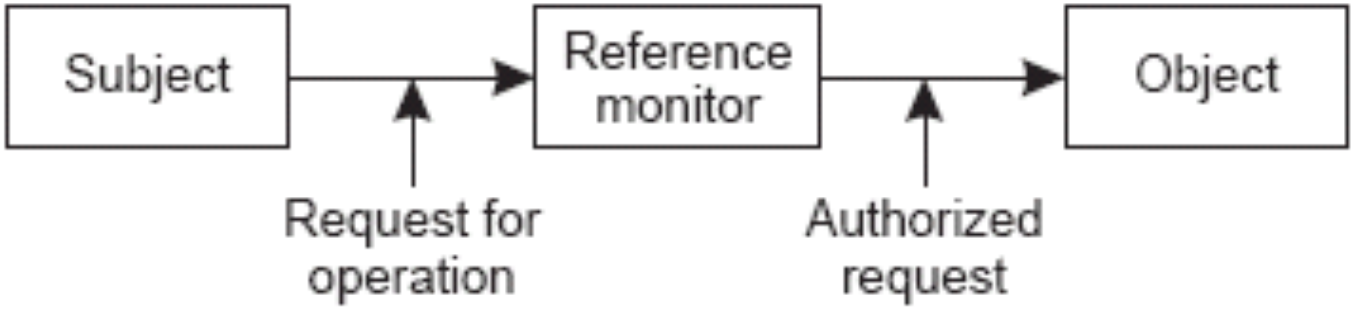
Access Control with Jess

CS3025, Knowledge-Based Systems
Lecture 19

Yuting Zhao
Yuting.zhao@gmail.com

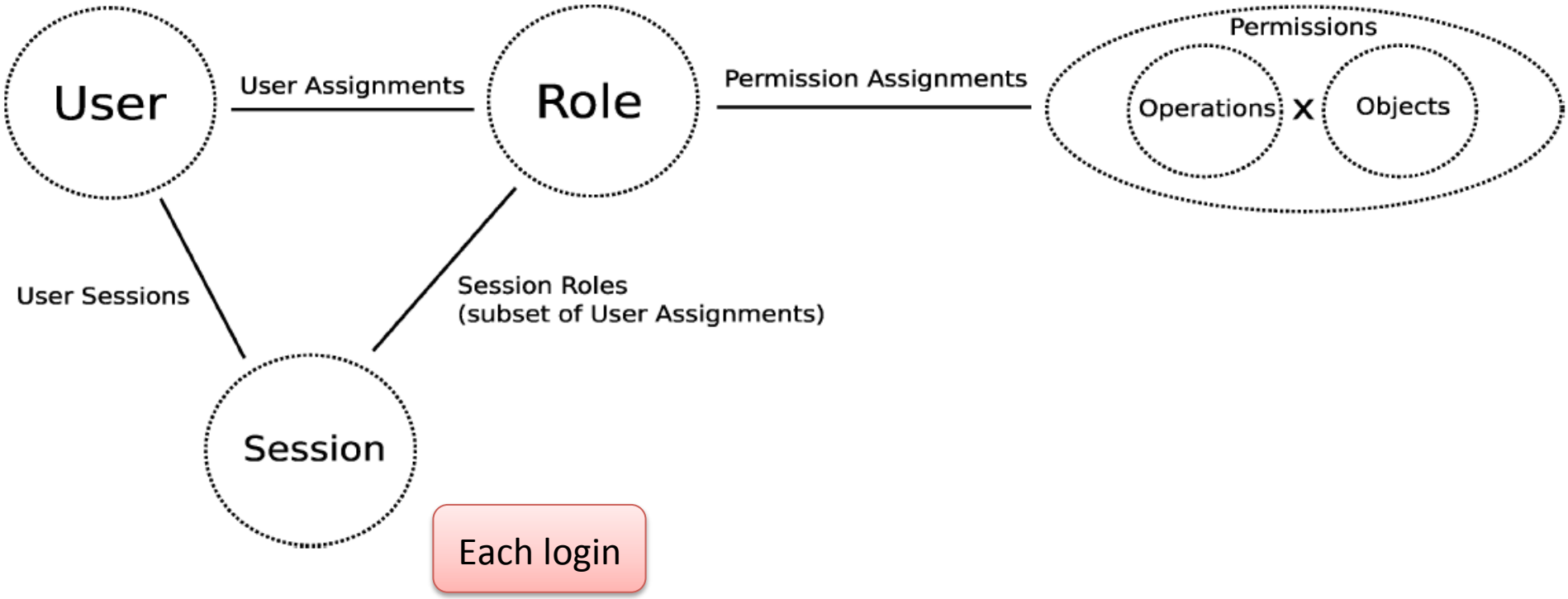
2017-11-21

The Basic Model for Access Control



- The ***subject*** (**user**) issues requests to access the ***object***, and protection is enforced by a ***reference*** *monitor* that knows which subjects are allowed to issue which requests

Role Based Access Control (RBAC)



RBAC User and Permission Assignments [*The RBAC American National Standard*, 2004]

Design RBAC & implementation by Jess

- RBAC Data model (facts)
  - user
  - session
  - role
  - grant
- Operations (rule)
  - allow
  - deny

- Questions about the lectures?

Questions related to the past Exam

- multi-agent systems, not covered now
-

Questions related to the past Exam

- Write a Jess function (using deffunction) to compute the distance between two points in a **Cartesian coordinate system**. The function should take four input parameters corresponding to those two points' coordinates, i.e., (x1, y1) and (x2, y2).
- Distance: $$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$
- Radical sign in Jess : (sqrt ?value)
- Code:

```
(deffunction distance (?X1 ?Y1 ?X2 ?Y2)
  (bind ?x (- ?X1 ?X2))
  (bind ?y (- ?Y1 ?Y2)))
  (bind ?distance (sqrt (+ (* ?x ?x) (* ?y ?y) )))
  (return ?distance))
```

Practice question

- Practice question: (**Exercise 20**)
- Write a function that takes as input a list of items and prints out this list in reverse order

Practice question

```
(bind ?input1(create$ AA BB CC DD EE ))
```

(AA BB CC DD EE)

```
(deffunction reverseorder (?input)
      (bind ?i (length$ ?input))
      (bind ?result (nth$ ?i ?input))
      (bind ?i (- ?i 1))

      (while (> ?i 0) do
            (bind ?newone (nth$ ?i ?input))
            (bind ?result ( create$ ?result ?newone  ))
            (bind ?i (- ?i 1))
      )
      return ?result
)

 (printout t " original: " ?input1 crlf " new order: " (reverseorder ?input1) crlf)
```