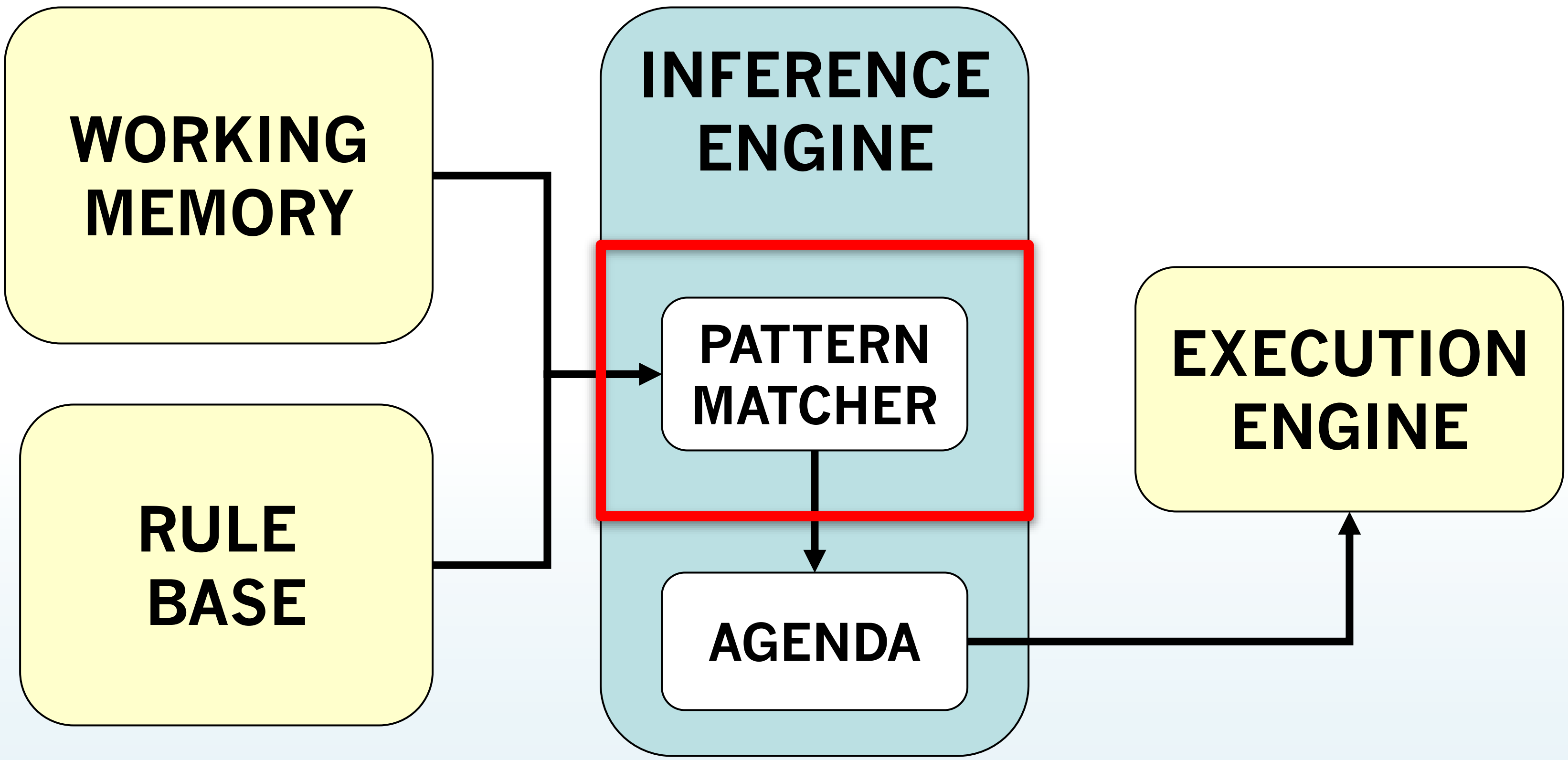
RBS


WORKING MEMORY
RULE BASE
INFERENCE ENGINE
PATTERN MATCHER
AGENDA
EXECUTION ENGINE

Inference with Production Rules

- There are two main issues involved in the implementation of a rule-based system
  - How is conflict resolution implemented?
    - Various strategies
  - How are facts matched to rules?
    - Pattern matching required
    - And many more issues need to be resolved

Inference Mechanisms

- Pattern matching and unification are powerful operations to determine the similarity and consistency of complex structures

- They are at the core of many rule-based and predicate logic mechanisms

- Their application goes beyond rule-based systems

Variables and bindings

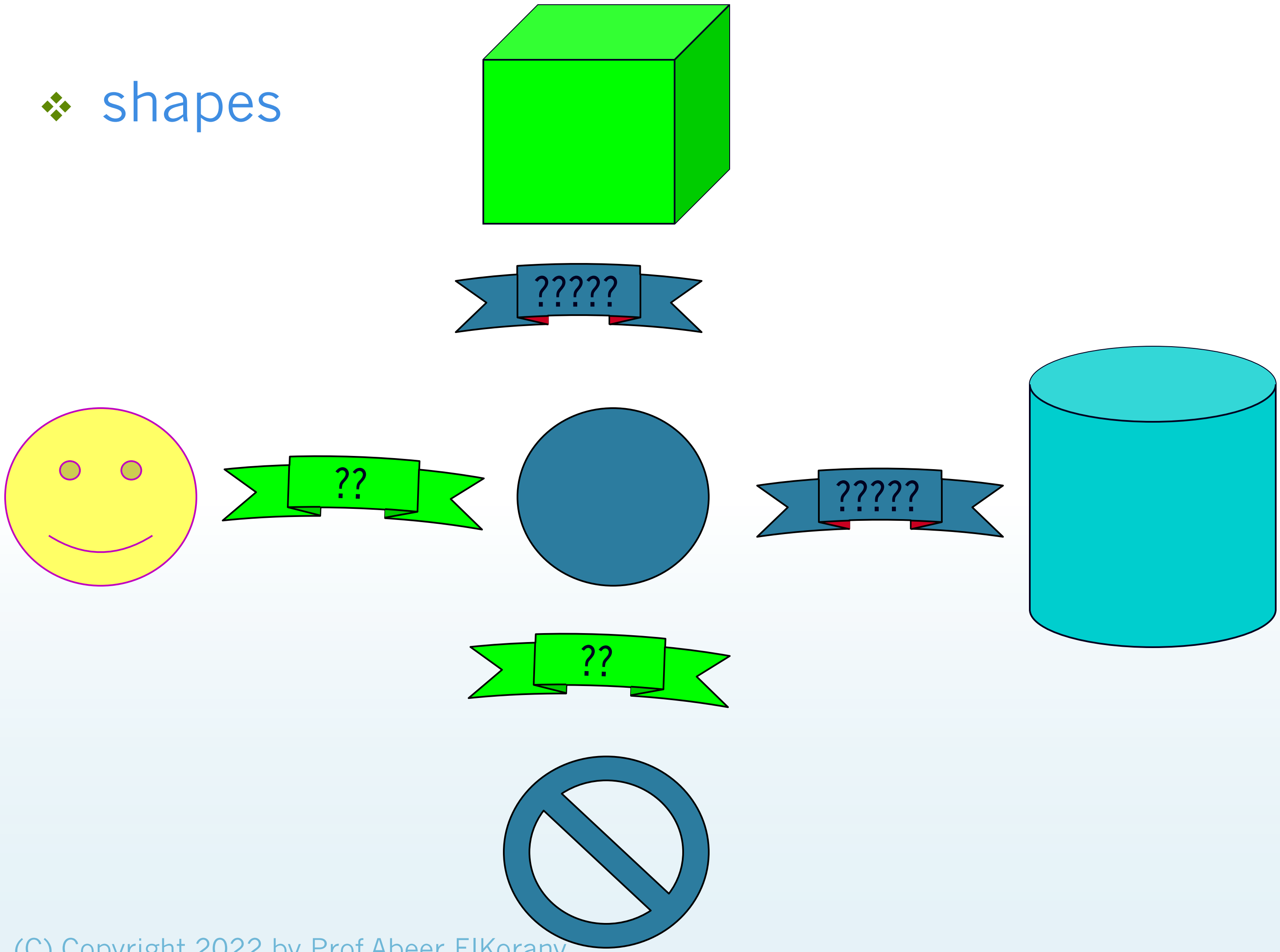
- Antecedents and consequents contain variables (?x)
- Variables acquire values during the matching process

Bindings

- Once a variable is bound, that variable is replaced by its binding wherever it appears in the same or subsequently processed patterns
- Whenever the variables in a pattern are replaced by their bindings, the pattern is said to be instantiated

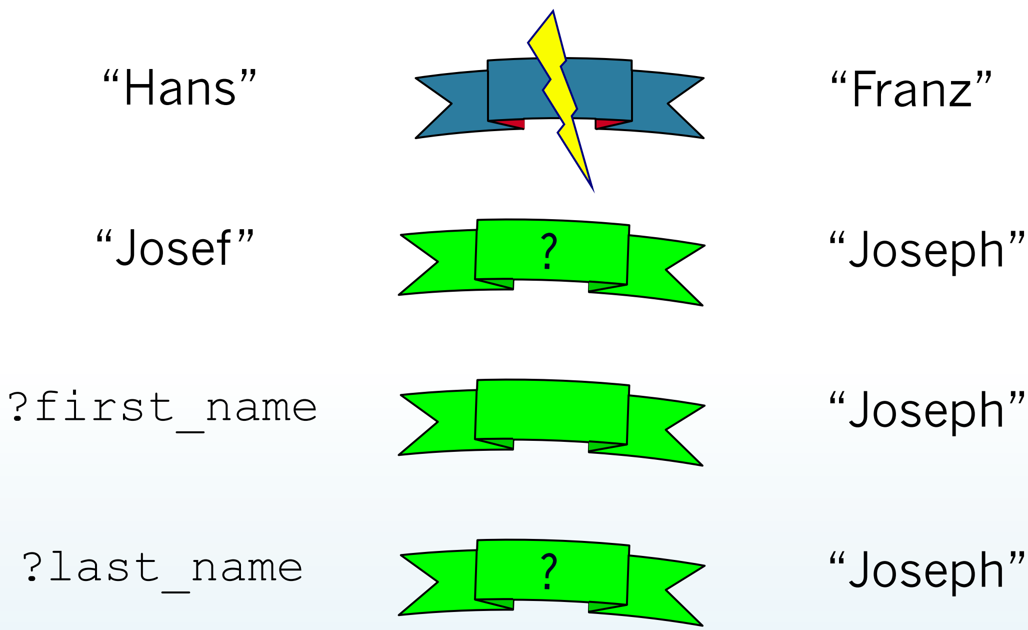
Pattern Matching Example

- shapes



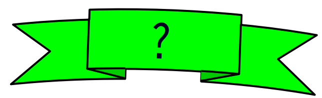
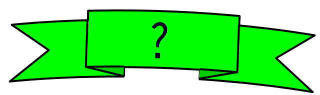
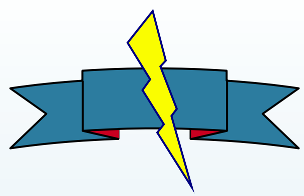
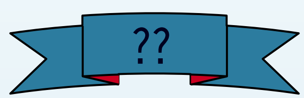
Pattern Matching Examples

- constants and variables



Pattern Matching Examples

- terms
  - composed of constants, variables, functions

`father(X)`		“Joseph”
`father(X)`		`father(Y)`
`father(X)`		`mother(X)`
`father(father(X))`		`grandfather(X)`

Applications of Pattern Matching search and retrieval

- Precise matching
  - strings, images, documents, objects
  - the query and the result have to match perfectly
  - the two should be identical
  - all requirements specified in the query have to be met by the result

- Similarity-based matching
  - imprecise

Pattern Matching in Rule-Based Systems

- Matching is the process of identifying which combinations of facts in the WM satisfy which rules

- Used to match rules with appropriate facts in working memory
  - rules for which facts can be found are *satisfied*
  - the combination of a rule with the facts that satisfy it is used to form *activation records*
    - one of the activation records is selected for execution
      - the respective rule "fires"

Search and Retrieval(cont.)

- similarity-based matching
  - requires specification of a similarity measure
  - often domain/application/task-specific
    - some aspects of the result are different from the query specification
      - but not too different
  - partial match
    - not all requirements specified in the query are met

Pattern Matching

?

- single field wildcard
- matches anything in corresponding field of fact

$?

- multi-field wildcard
- matches zero or more fields of a fact

Pattern Matching (cont.)

?<*var*>

- single field variable
- <var> is some word
- this symbol matches anything in the corresponding field of a fact
- value matched is bound to ?<*var*> for scope of rule
- examples: ?cat ?color ?machine

$?<*var*>

- multi-field variable
- matches zero or more fields of a fact
- the value(s) of the matched fields is bound to $?<*var*> for the scope of the rule

Examples

Single field wild cards

LHS Condition	Fact in Fact Base	Match?
(? ?)	(data red)	Yes
(data ?)	(data red)	Yes
(data ?)	(data red green)	NO!
(data ? ?)	(data red green)	Yes
(data red ?)	(data red green)	Yes
(data ? green)	(data green)	NO!

Examples

Single Field Variables

LHS Condition	Fact in Fact Base	Match?
(data red ?x)	(data red green)	?x=green
(data red ?x)	(data red “green”)	?x=“green”
(data red ?x)	(data red 17.4)	?x=17.4
(data ?x ?x)	(data red red)	?x=red
(data ?x ?x)	(data red green)	NO!
(data ?x ?y)	(data red green)	?x=red ?y=green
(data ?x ?y)	(data red red)	?x=red ?y=red

Examples

Multi-field wild cards

LHS Condition	Fact in Fact Base	Match?
($?)	(data red)	Yes
(data $?)	(data red)	Yes
(data red $?)	(data red)	Yes
(data $?)	(data red green)	Yes
(data red green $?)	(data red green)	Yes
($? green)	(data red green)	Yes
($? red $?)	(data red green)	Yes
(data red $?)	(data green red)	NO!
(data $? red $?)	(data green red)	Yes
($? $?)	(data red)	Yes

Examples

Multi-Field Variables

LHS Condition	Fact in Fact Base	Match?
(data red $?x)	(data red)	$?x=()
(data red $?x)	(data red green)	$?x=(green)
(data red $?x)	(data red one two)	$?x=(one two)
(data $?x $?x)	(data red red)	$?x=(red)
(data $?x $?y)	(data red green)	Multiple matches:

$?x=()	$?y=(red green)
$?x=(red)	$?y=(green)
$?x=(red green)	$?y=()

Rule-Based Pattern Matching

- Go through the list of rules, and check the antecedent (LHS) of each rule against the facts in working memory
  - create an activation record for each rule with a matching set of facts
  - repeat after each rule firing
- Very inefficient
  - roughly (number of rules) * (number of facts)
  - the actual performance depends on the formulation of the rules and the contents of the working memory

Disadvantages of Rule Systems

- Require exact matching
  - IF The motor is hot
  - THEN Shut the motor down
  - What about
    - The motor is running hot.
    - The motor's temperature is hot.

- Have opaque rule relationship(difficult to debug for a large rule base)
  - IF C THEN D
  - IF B THEN C
  - IF A THEN B

- Can be slow (May need to scan the whole rule set several times)

Problems with Inference cycle

1. Complete scan of the working memory (facts list) during each cycle of execution
2. More rules will result in more rules that apply and more facts being derived that both apply and do not apply

- Suppose you have the following situation:
  - – *r* rules in KB with average of *p* conditions
  - – *f* facts in the fact base
- – Then our inference engine will have to perform roughly $r*f^{p}$ comparisons each cycle

Match step

- Disadvantage of RBS – large computational requirement to perform match of LHS – determine if all instantiations of rules are satisfied by the content of the WM (Working Memory)

- **O**(comparison to check the satisfaction of a rule) **= $|WM|^{|CE|}$**

|WM| - no of WMEs (Working Memory Elements)

|CE| - number of condition elements in a rule

Rule example

```
(deftemplate  student
      (slot name)
      (slot gender)
      (slot placed_in (default nil))
      (slot special_considerations
                  (default no))
```

```
(deffacts start
(student (name Mary) (gender F))
(student (name Peter) (gender M))
      (special_consideration yes))
(room (number 221) (capacity 1) (vacancies 1))
(room (number 346) (capacity 2) (vacancies 1)))
```

```
(deftemplate room
      (slot number)
      (slot capacity (type INTEGER)(default 4))
      (slot genderes_are)
      (slot vacancies (type INTEGER))
      (multislot occupants))
```

WMEs
41 (student name Mary gender F placed_in nil
special_consideration no)
52 (student name Peter gender M placed_in nil
special_consideration yes)
9 (room number 221 capacity 1 vacancies 1)
12 (room number 346 capacity 2 vacancies 1)

Rule example

```
(defrule  assign-student-empty-room
     ?unplaced_student ← (student (name  ?stud_name)
                                  (placed_in nil)
                                  (gender  ?gen))
     ?empty_room ← (room (number ?room_no)
                         (capacity ?room_cap)
                         (vacancies  ?free_places))
     (test  (=   ?room_cap    ?free_places))
=>
     (modify  ?unplaced_student  (placed_in ?room_no))
     (modify  ?empty_room  (occupants  ?stud_name) (gender_are ?gen)
                         (vacancies (-- ?free_places))))
```

$\forall s \forall r \forall x \forall y \forall z \forall t$ name(s,x) $\wedge$ placed_in(s,nil) $\wedge$ gender(s,y) $\wedge$ number(r,z) $\wedge$ capacity(r,t) $\wedge$ vacancies(r,t) $\rightarrow$ placed_in(s,z) $\wedge$ ....

Making it more efficient

- The **Rete algorithm** is an efficient pattern matching algorithm for implementing rule-based expert systems.
- The Rete algorithm was designed by Dr. Charles L. Forgy of Carnegie Mellon University in 1979.
- the name comes from the latin word *rete*
  - stands for net
- Rete has become the basis for many popular expert systems, including OPS5, CLIPS, and JESS.
- The Rete algorithm was the first efficient solution to the facts-rules pattern matching problem
  - It stores information about matches in a network structure

Rete Algorithm

- Rete looks for changes to match in each cycle
- Focus on few facts that are added, changed or removed at every step in the process of inference. Instead of doing all these comparisons every time only new facts added can be taken into consideration.
- Unnecessary Computations Facts Change (Keep track of changes) Rules remain unchanged
- the Rete algorithm performs an improved matching of rules and facts
  - basis for many rule-based expert system shells

The Rete Matching Algorithm

- Nodes of the network correspond to individual condition elements
  - Conditions and conjunctions of conditions
- Each node has two sets associated with it
  - The first set contains all the working memory elements that the condition node matches
  - The second set contains combinations of working memory elements and the bindings which produce a consistent match of the conditions that chain up to the node condition

RETE algorithm

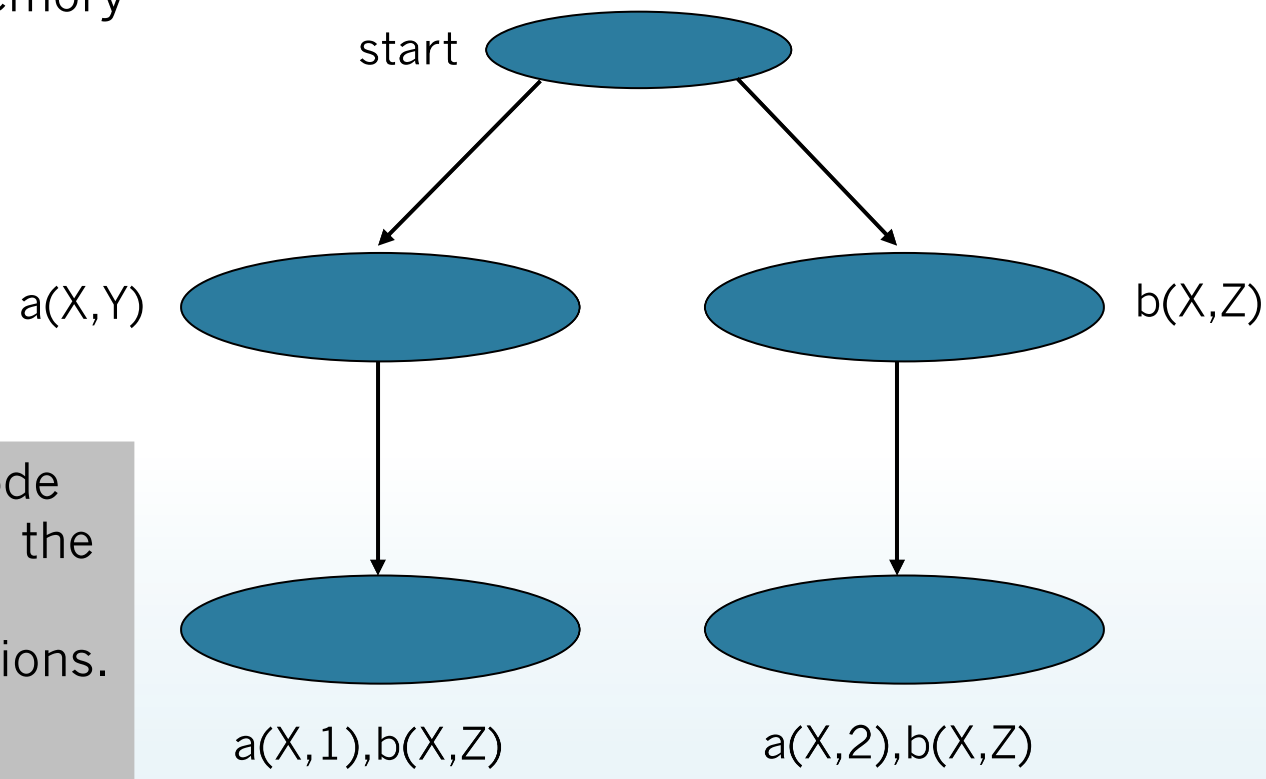
- Consists of a network of interconnected nodes
- Creates a decision tree where each node corresponds to a pattern occurring at the left-hand side of a rule
- Each node has a memory of facts that satisfy the pattern
  - input nodes are at the top, output nodes at the bottom
  - *join nodes* have two inputs, and combine facts
  - *terminal node* at the bottom of the network represent individual rules
- a rule is satisfied if there is a combination of facts that passes all the test nodes from root to a leaf.

The Rete Matching Algorithm

- With this configuration repetitive testing of all rule conditions in each cycle is avoided
  - Only the nodes affected by a newly inserted or modified fact are checked
  - For example, consider the rules
    - IF a(X,1) and b(X,Z) THEN g1(X,Z)
      IF a(X,2) and b(X,Z) THEN g2(X,Z)

The Rete Matching Algorithm

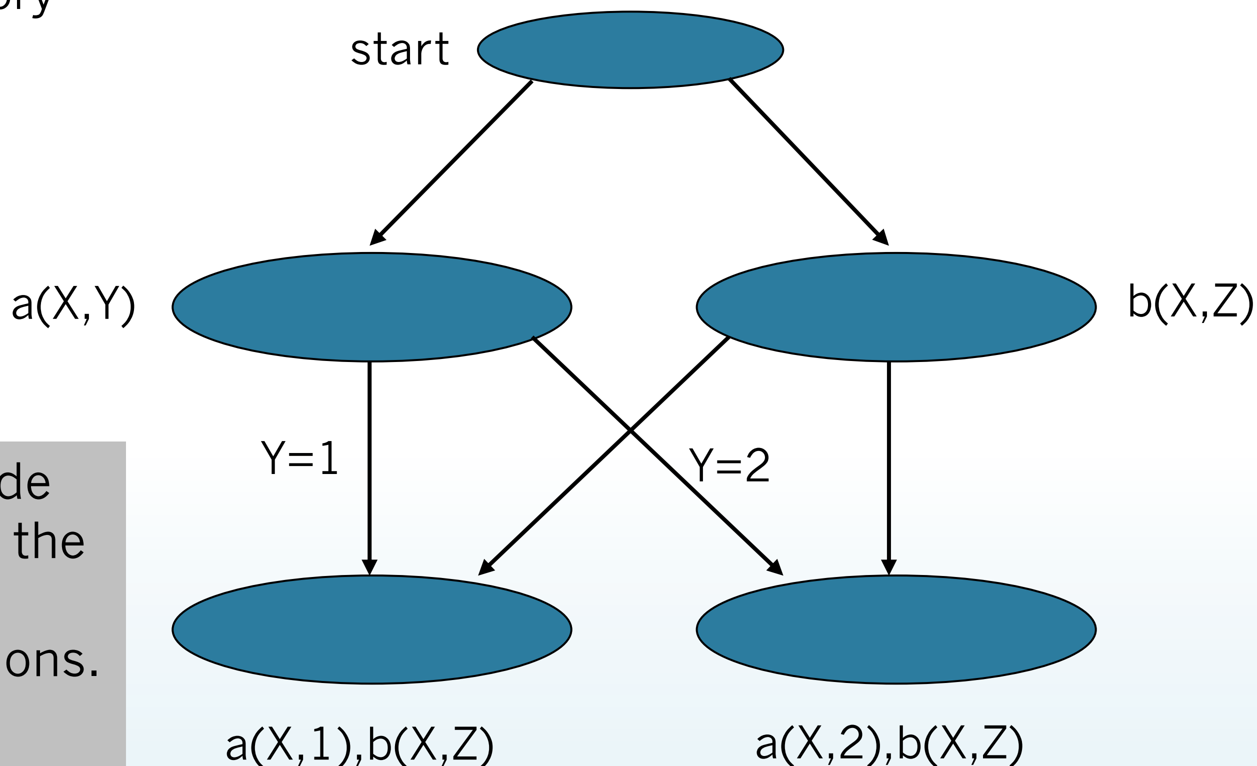
Initially the working memory is empty



- There is a starting node and a node for each of the rule conditions and conjunctions of conditions.
- Arcs are labeled with variable bindings

The Rete Matching Algorithm

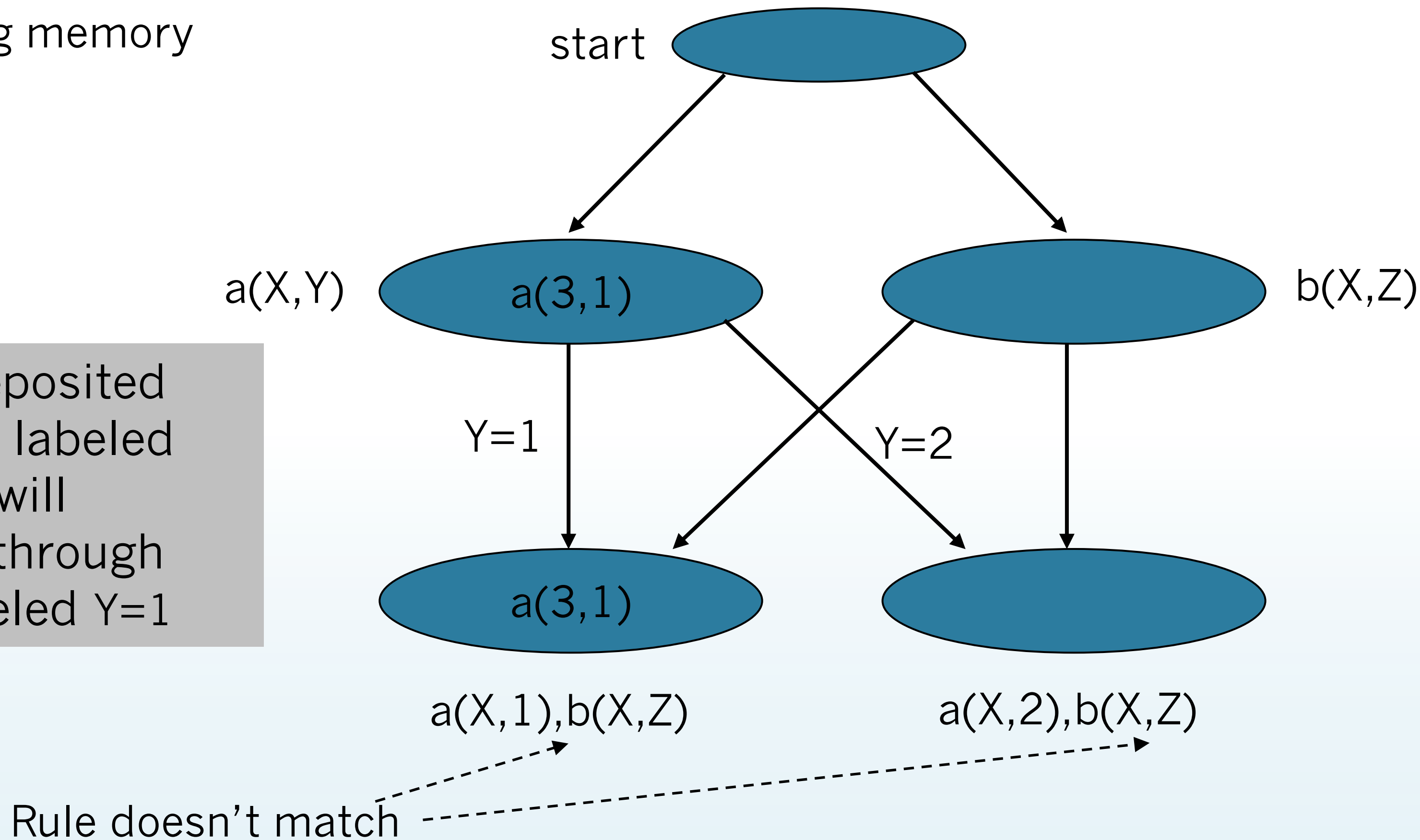
Initially the working memory is empty



- There is a starting node and a node for each of the rule conditions and conjunctions of conditions.
- Arcs are labeled with variable bindings

The Rete Matching Algorithm

Fact a(3,1) is added to the working memory

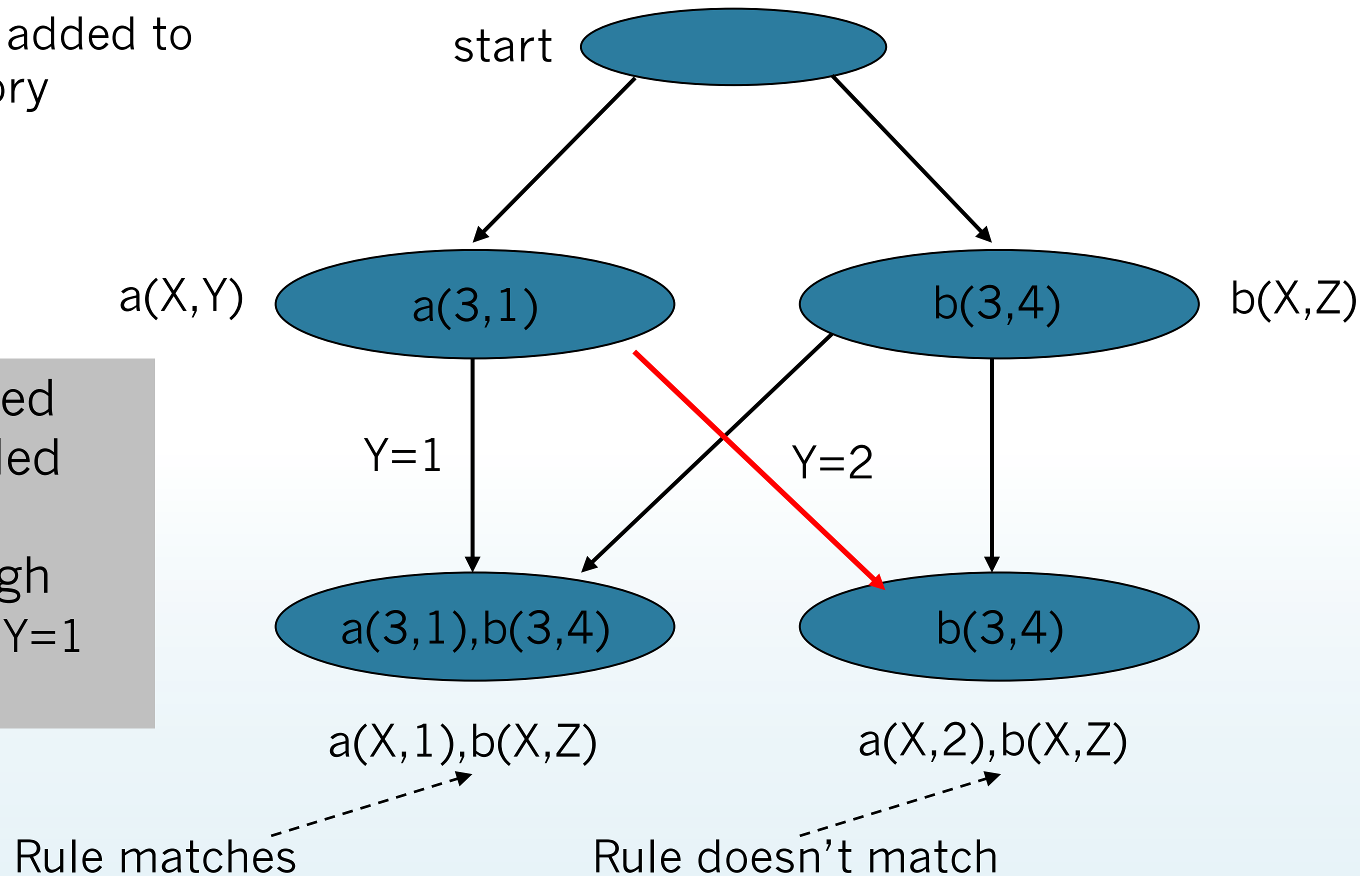


a(3,1) is deposited in the node labeled a(X,Y) and will propagate through the arc labeled Y=1

The Rete Matching Algorithm

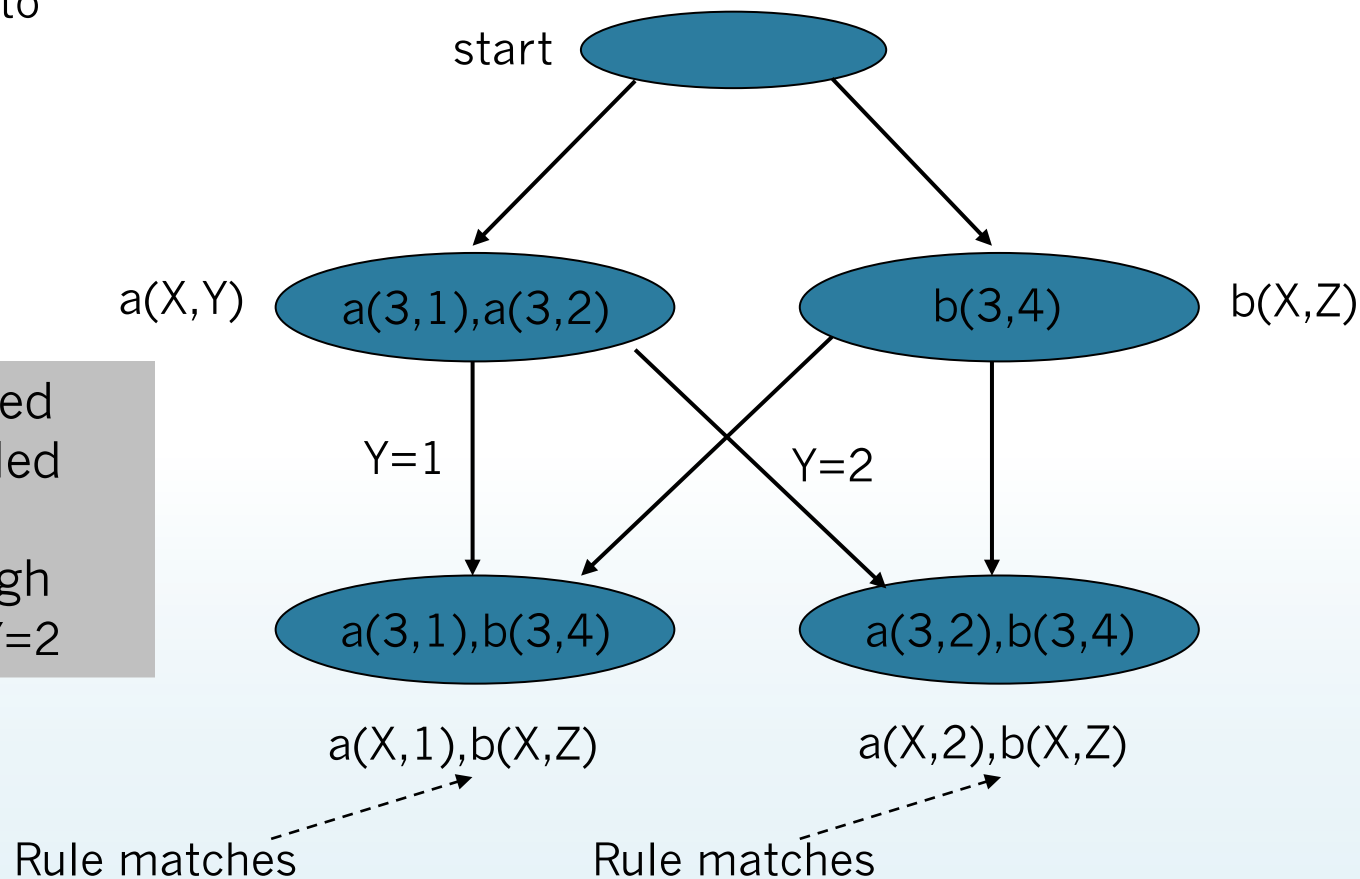
Fact a(3,1)
Fact b(3,4) is now added to the working memory

b(3,4) is deposited in the node labeled b(Y,Z) and will propagate through the arcs labeled Y=1 and Y=2



The Rete Matching Algorithm

Fact a(3,1)
Fact b(3,4)
Fact a(3,2) is added to the working memory



a(3,2) is deposited in the node labeled a(X,Y) and will propagate through the arc labeled Y=2

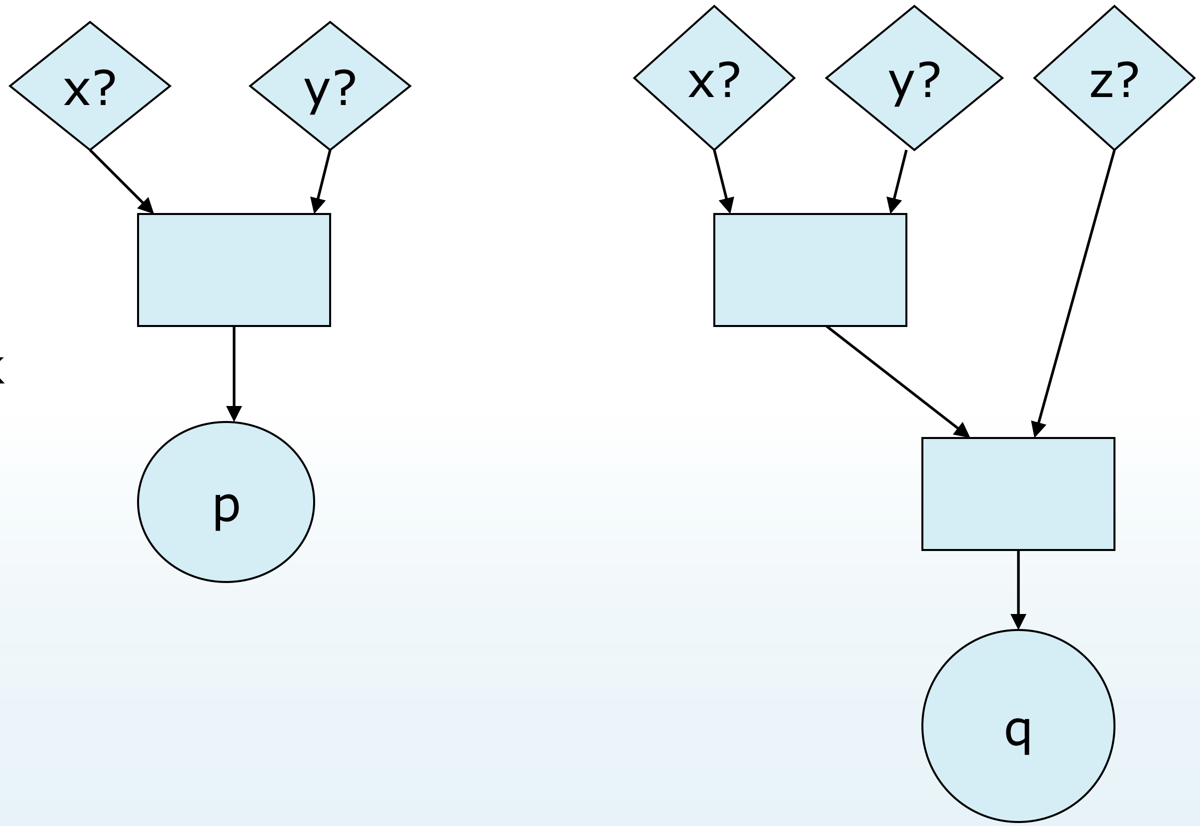
Rete example

R1: IF x & y THEN p
R2: IF x & y & z THEN q

Pattern Network

Join Network

8 nodes



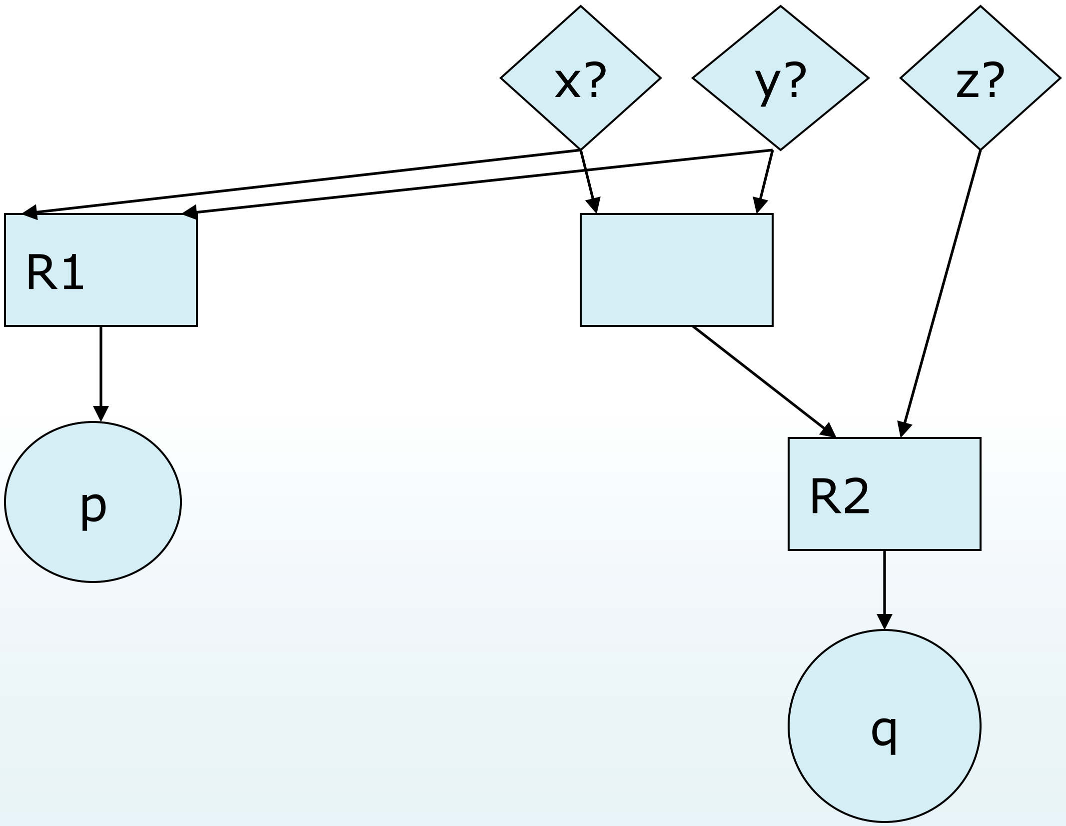
Rete example

R1: IF x & y THEN p
R2: IF x & y & z THEN q

Pattern
Network

Join Network

6 nodes



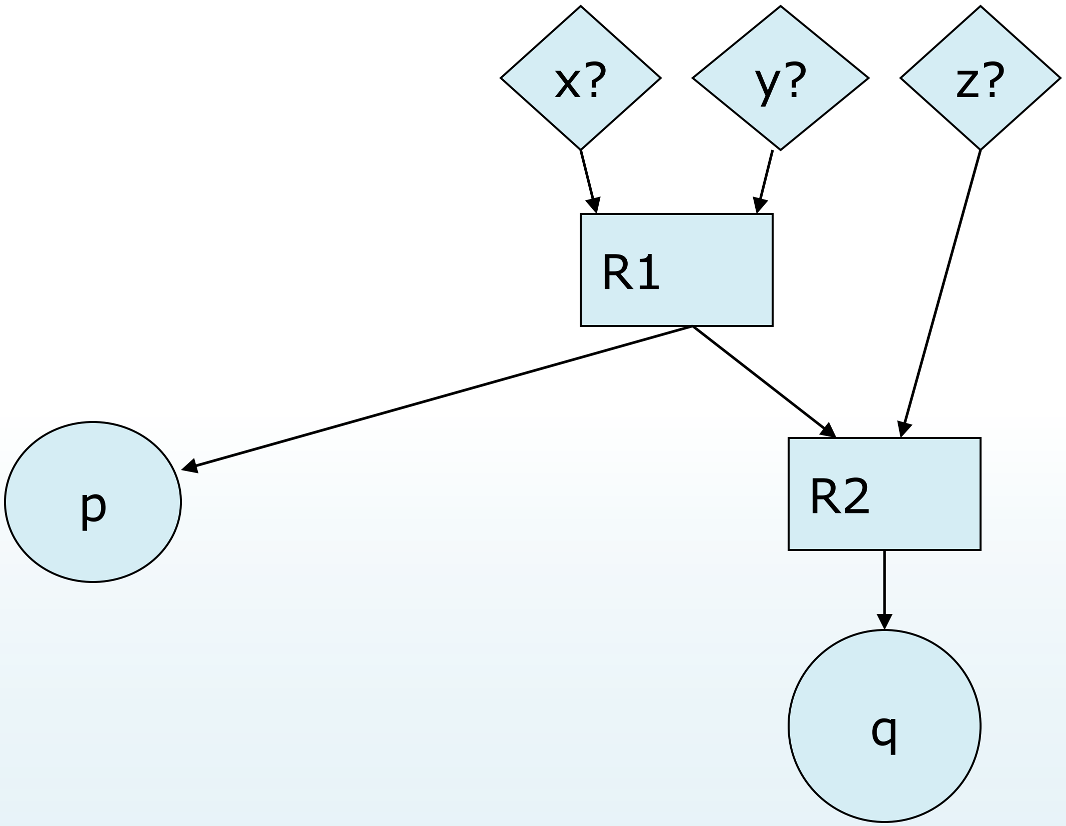
Rete example

R1: IF x & y THEN p
R2: IF x & y & z THEN q

Pattern
Network

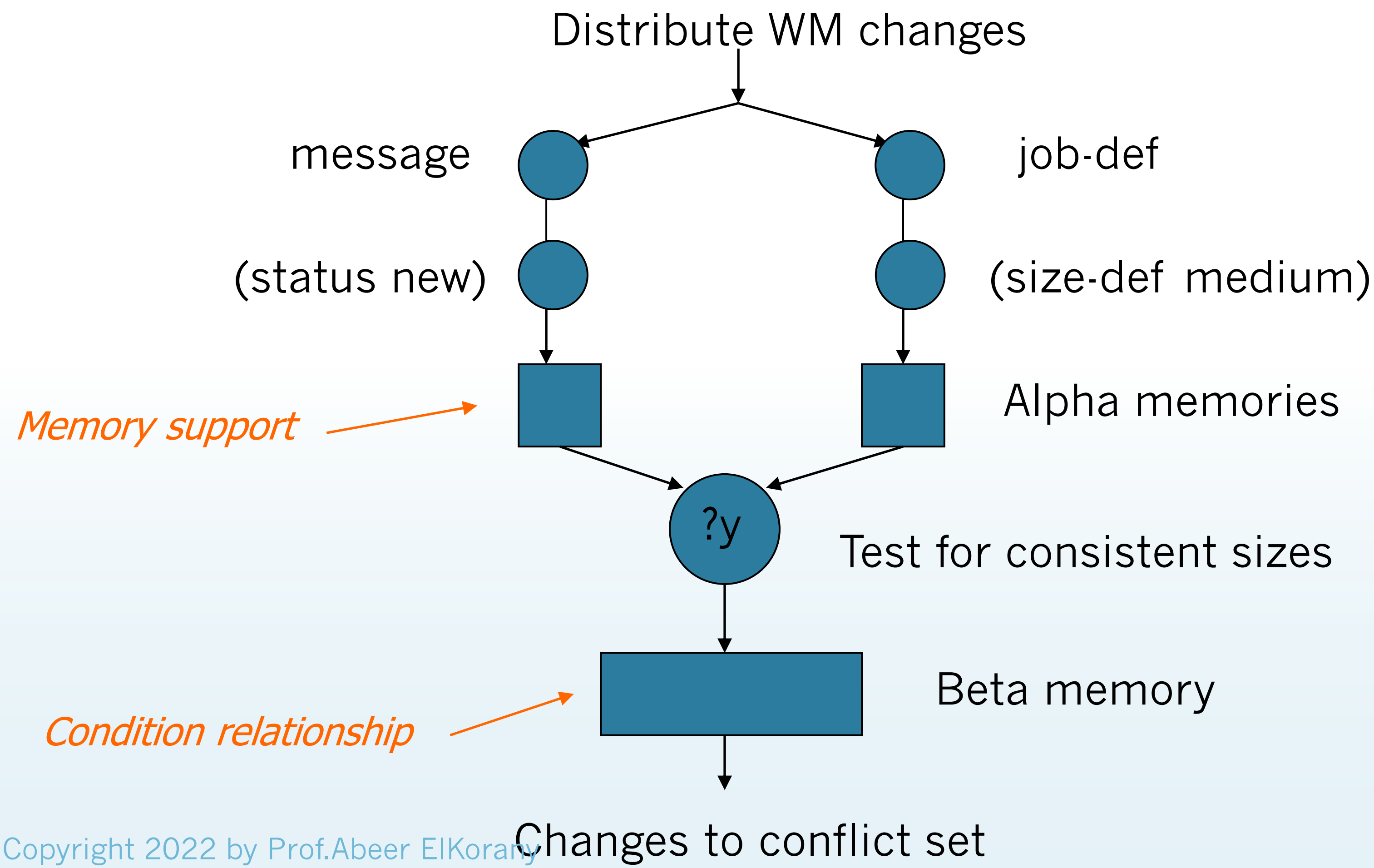
Join Network

5 nodes



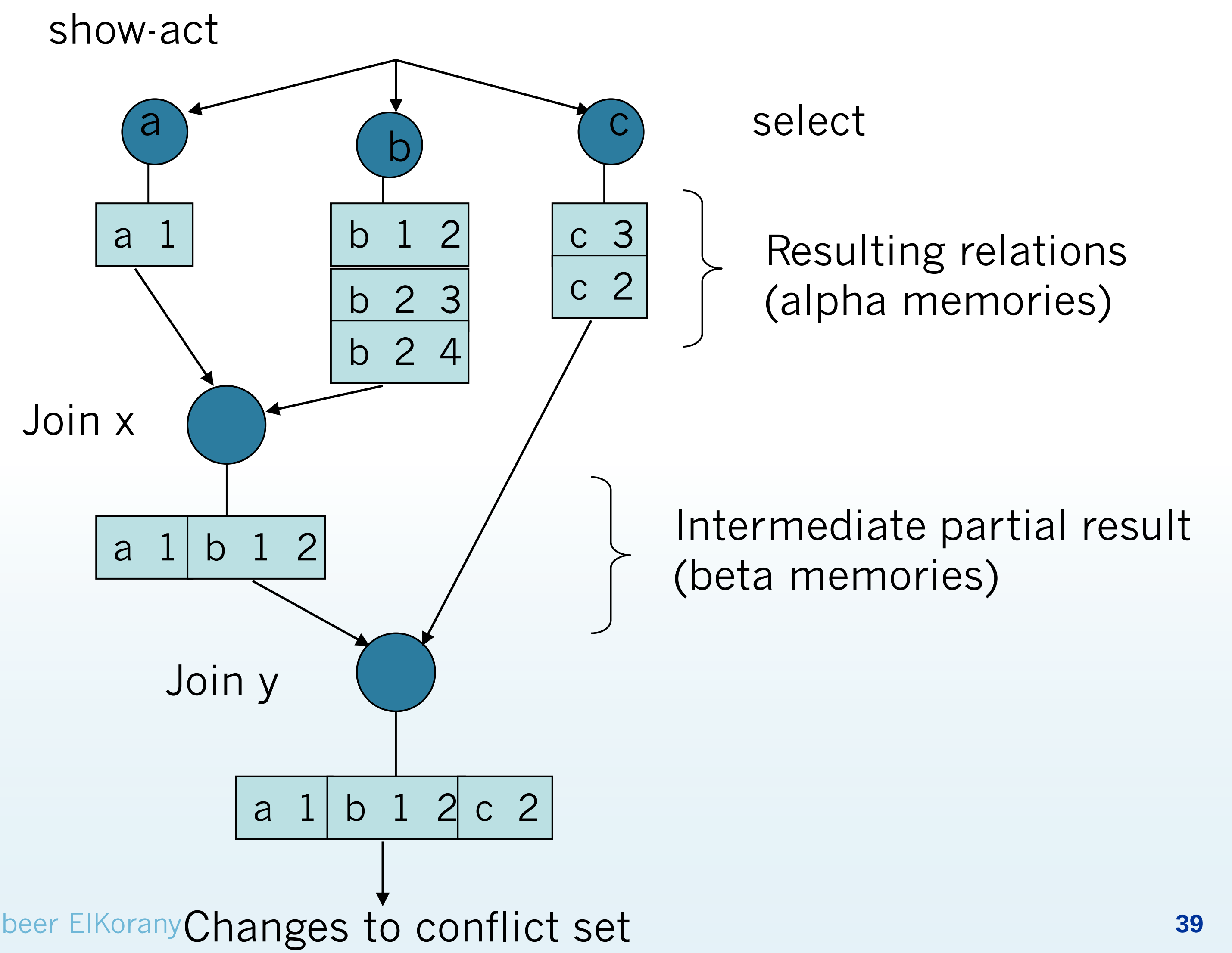
```
(defrule job-size
     (message (jobs ?x) (size ?y) (status new))
     (job-def (size ?y) (size-def medium))
=>
    (assert (job (job-name ?x) (job-size ?y))))
```

RETE match Network for the rule



```
(defrule  show-act              WM
     (a  ?x)                    (a   1)
     (b  ?x  ?y)                (b   1  2) (b   2  3)  (b  2  4)
     (c  ?y )                   (c   3) (c  2)
=> …
```

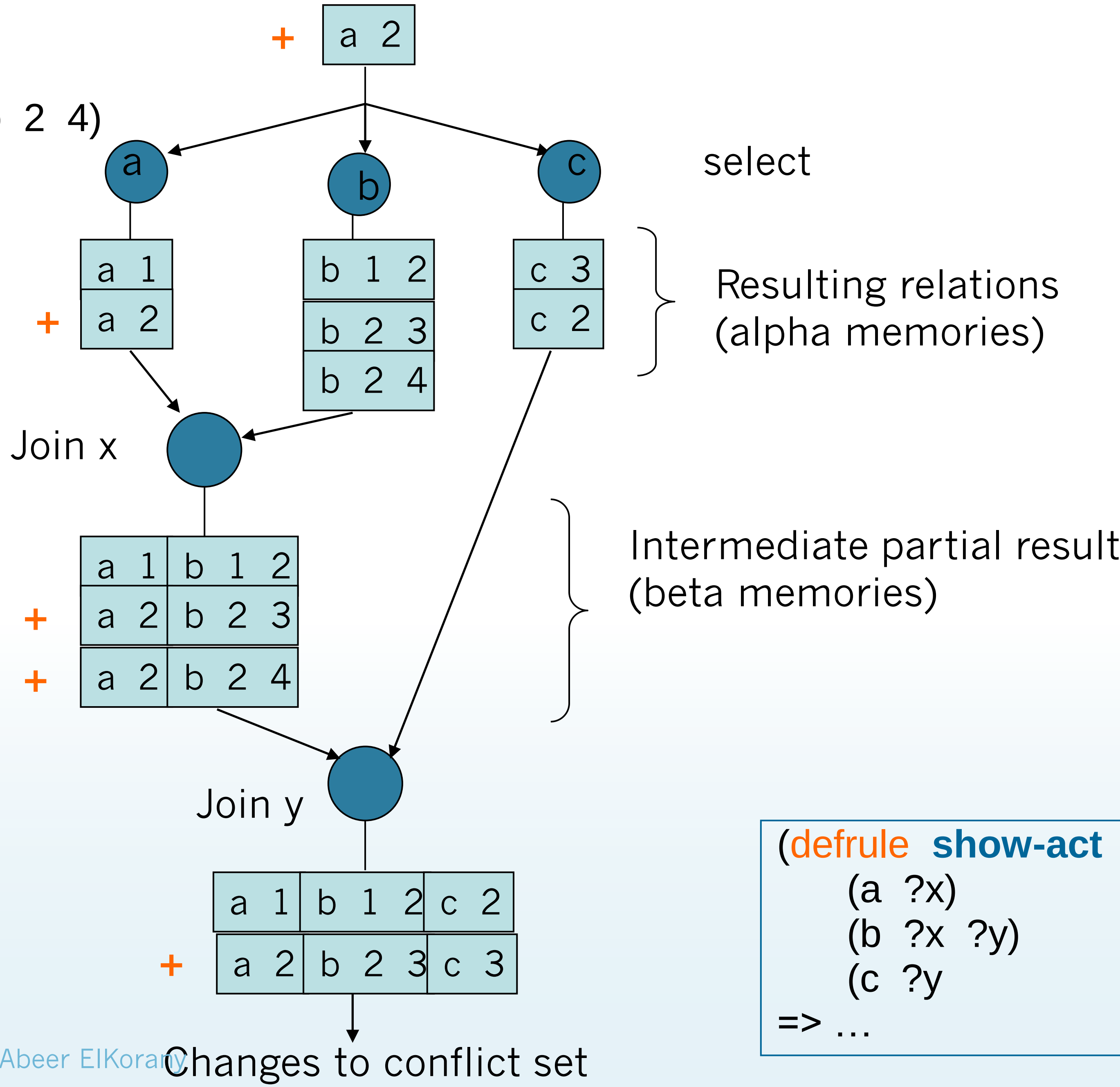
Initial state of RETE Network



Activity of the RETE Match during an Addition

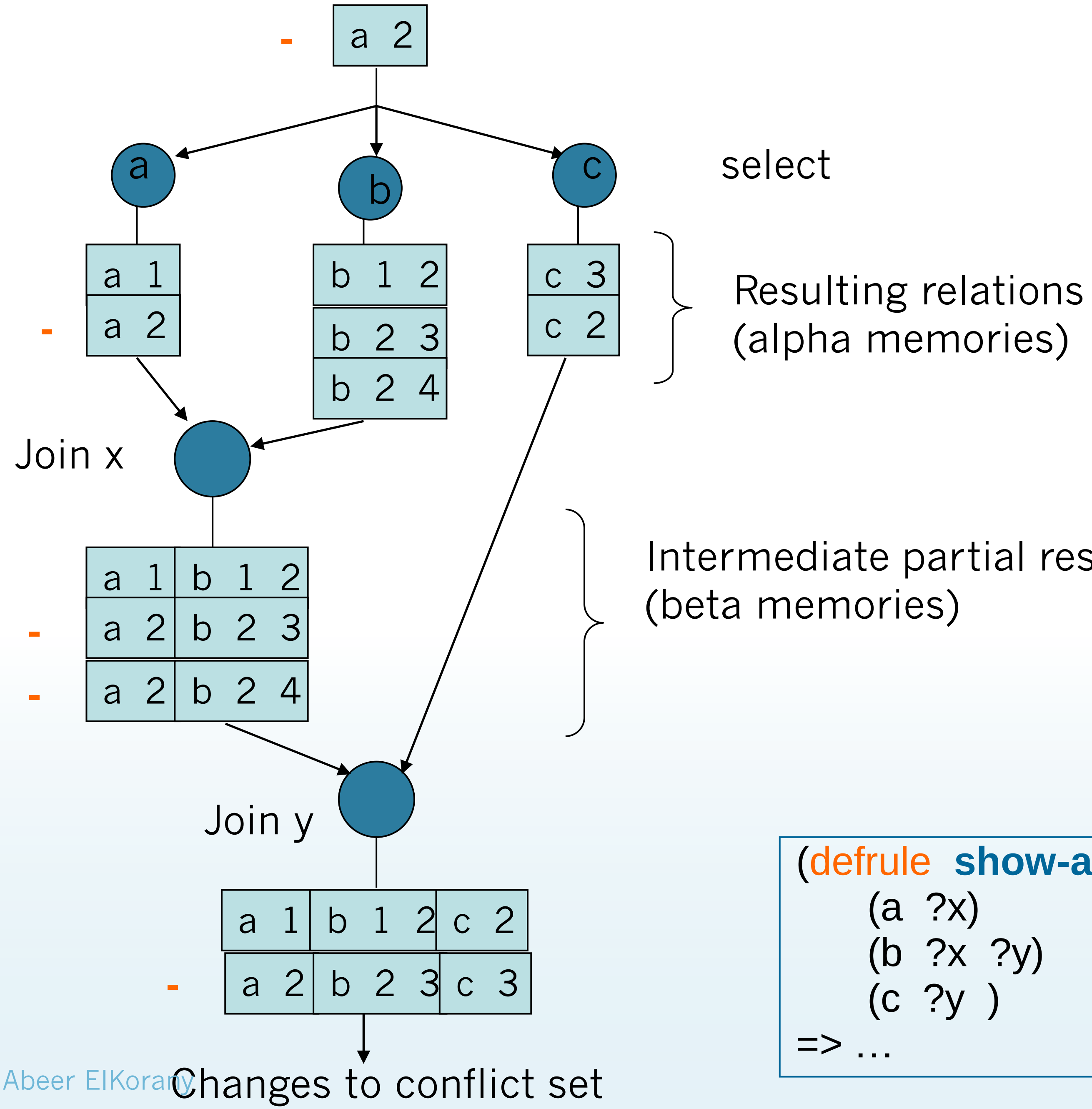
WM

(a 1)
(b 1 2) (b 2 3) (b 2 4)
(c 3) (c 2)
(a 2)




```
(defrule show-act
    (a ?x)
    (b ?x ?y)
    (c ?y
=> …
```

Activity of the RETE Match during a Deletion



(defrule show-act
    (a ?x)
    (b ?x ?y)
    (c ?y)
=> …

Assert and Retract with Rete

- asserting additional facts imposes some more constraints on the network
- retracting facts indicates that some previously computed activation records are not valid anymore, and should be discarded
- in addition to the actual facts, *tags* are sent through the networks
  - `ADD` to add facts (i.e. for `assert`)
  - `REMOVE` to remove facts (i.e. for `retract`)
  - `CLEAR` to flush the network memories (i.e. for `reset`)
  - `UPDATE` to populate the join nodes of newly added rules
    - already existing join nodes can neglect these tokens

Further Optimizations

- sophisticated data structures to optimize the network
  - hash table to presort the tokens before running the join node tests
- fine-tuning via parameters
  - frequently trade-off between memory usage and time

Further Optimizations

Example:

1 IF Shape=long
& Color=(green or yellow)
THEN Fruit=banana

2 IF Shape=(round or oblong)
& Diam>4
THEN Fruitclass=vine

3 IF Shape=round & Diam<4
THEN Fruitclass=tree

4 IF Seedcount=1
THEN Seedclass=stonefruit

5 IF Seedcount>1
THEN Seedclass=multiple

6 IF Fruitclass=vine & Color=green
THEN Fruit=watermelon

7 IF Fruitclass=vine
& Surface=smooth & Color=yellow
THEN Fruit=honeydew

8 IF Fruitclass=vine &
Surface=rough
& Color=tan THEN Fruit=cantaloupe

9 IF Fruitclass=tree & Color=orange
& Seedclass=stonefruit & Diam>3
THEN Fruit=peach

10 IF Fruitclass=tree & Color=orange
& Seedclass=multiple
THEN Fruit=orange

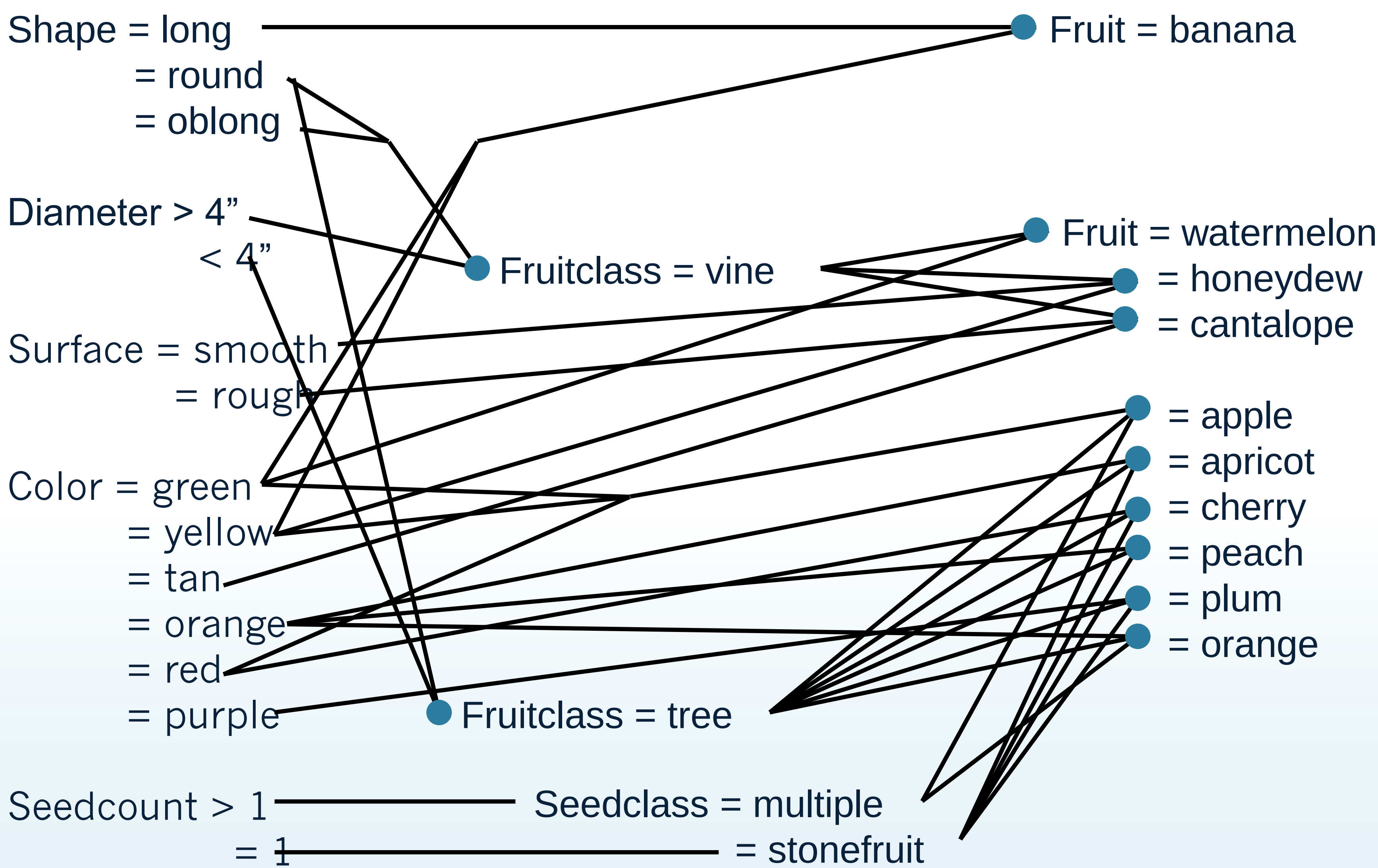
11 IF Fruitclass=tree & Color=red
& Seedclass=stonefruit
THEN Fruit=cherry

12 IF Fruitclass=tree & Color=orange
& Seedclass=stonefruit & Diam < 3
THEN Fruit=abricot

13 IF Fruitclass=tree
& Color=(red or yellow or green)
& Seedclass=multiple
THEN Fruit=apple

14 IF Fruitclass=tree & Color=purple
& Seedclass=stonefruit
THEN Fruit=plum

Fruit Knowledge Base Example



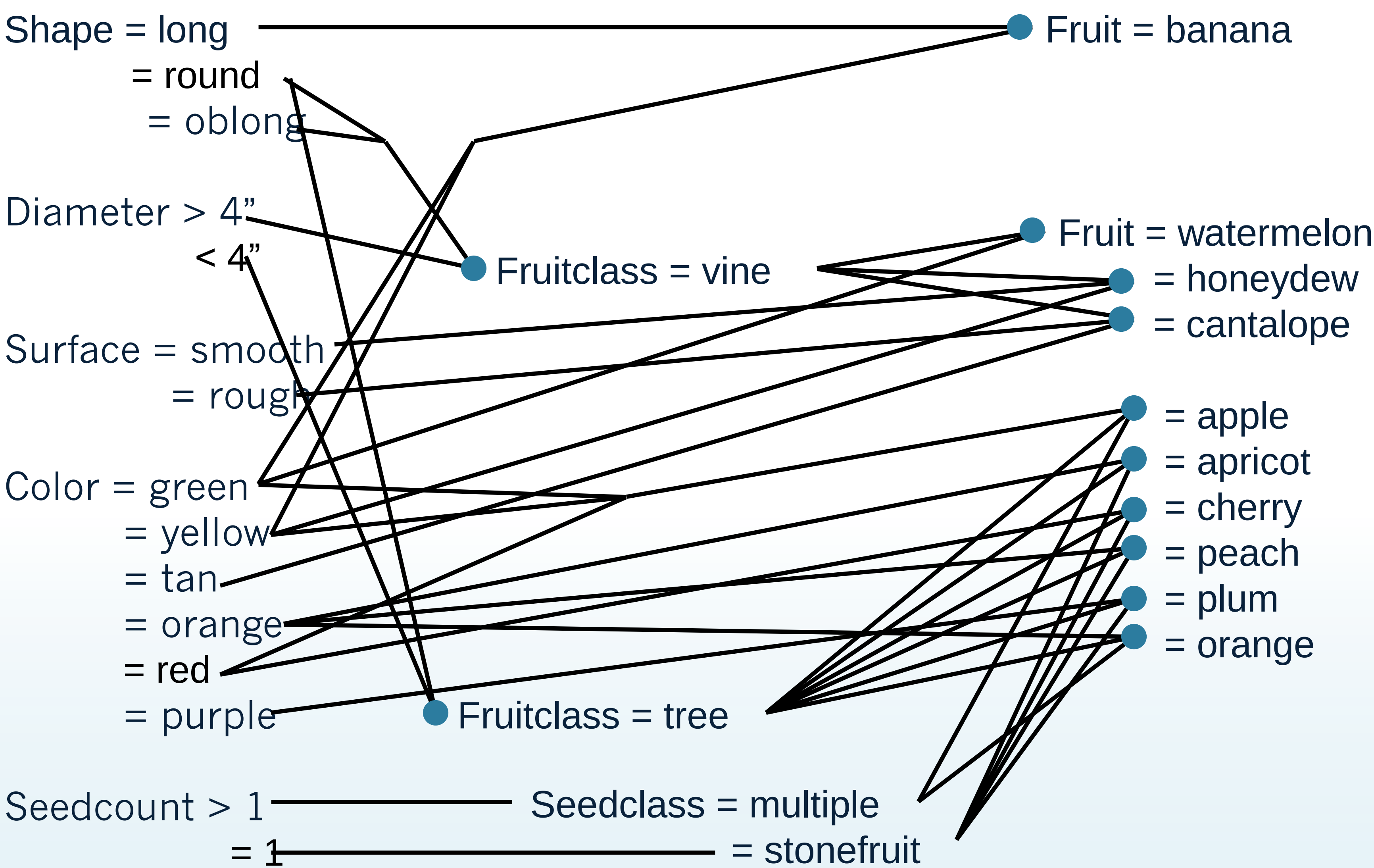
Sample Facts

```
(deffact list
    (color is red)
     (shape is round)
     (= seedcount 1)
     (< diameter 4)
    ) )
```

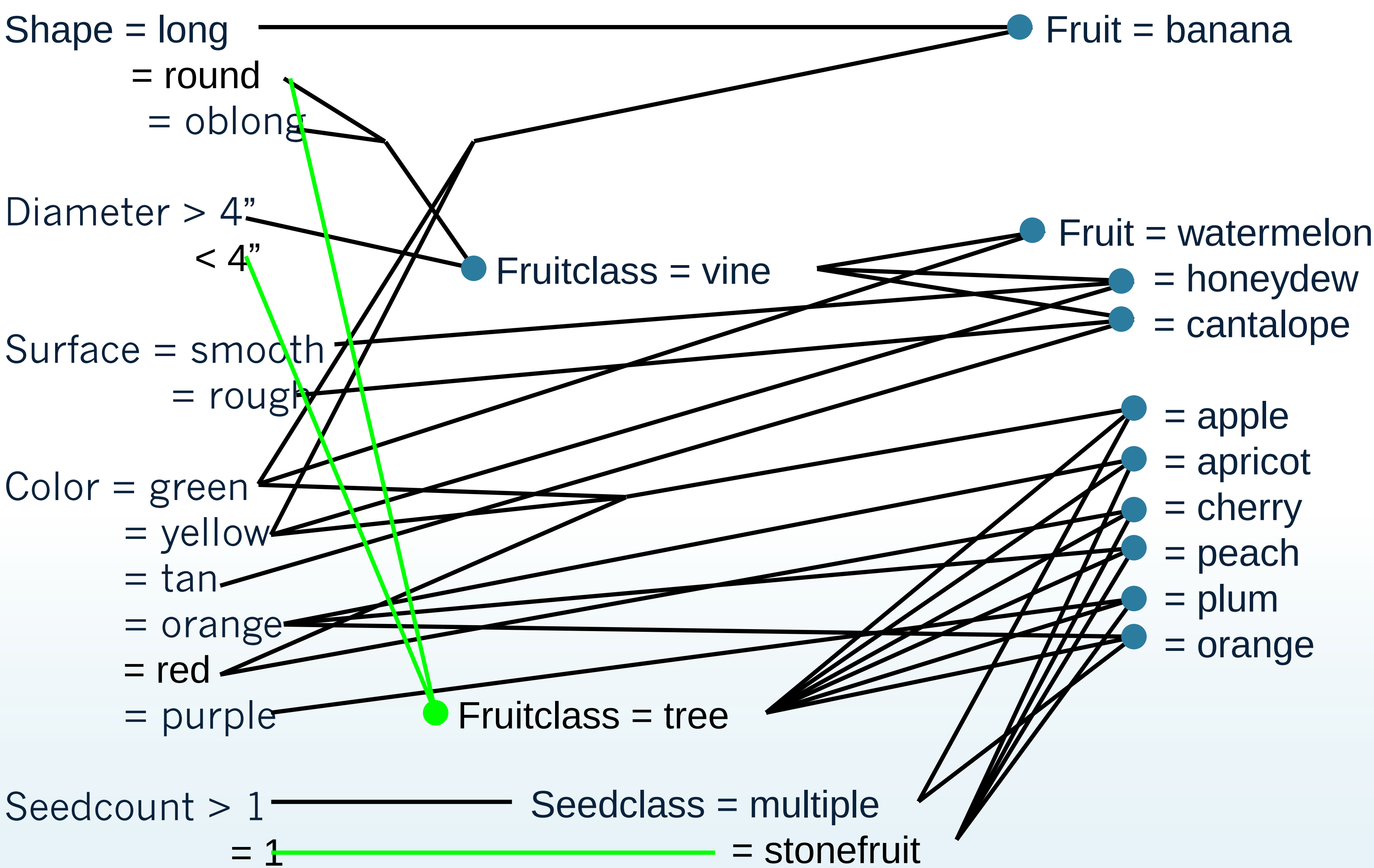
What is the result from execution?

The value of fruitclass is tree
The value of seedclass is stonefruit
The value of fruit is cherry

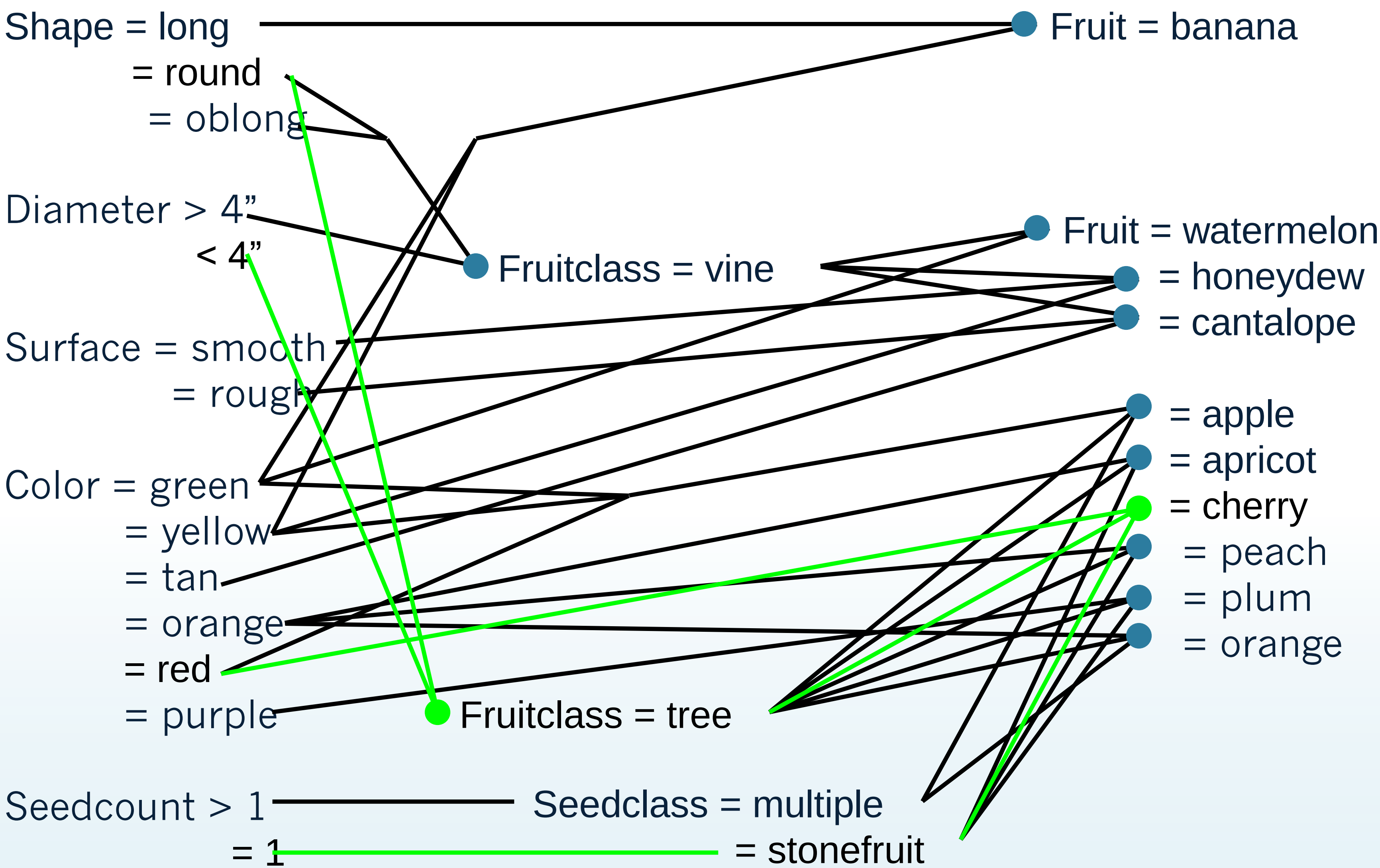
Fruit Knowledge Base



Fruit Knowledge Base

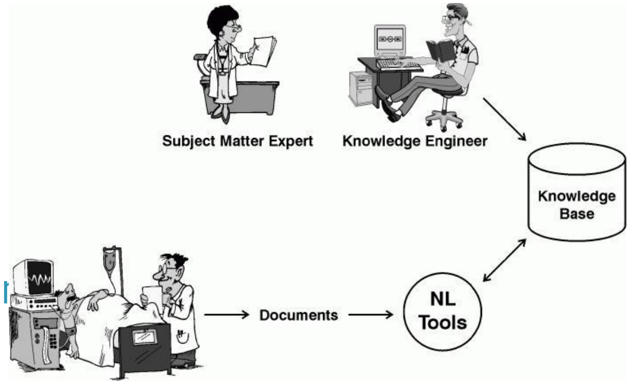


Fruit Knowledge Base



Designing efficient rule-based systems

- Knowledge Engineer
  - Tasked with working with the expert to extract expertise and codify in a set of rules
  - Has training in the development of expert systems, but n necessarily in the application domain
  - Know the capabilities of technology and knows how to apply it



- The most specific pattern should be placed toward the front of the left-hand side of the rule
- A specific pattern will generally have the **smallest** number of matching facts in the facts list and will have the **largest** number of variable bindings which constrain other patterns.

Most specific pattern goes first

- R1: (item ?x) (item ?y) (item ?z) (match ?x ?y ?z))-> (buy ?x)
- R2 : (match ?x ?y ?z) (item ?x) (item ?y) (item ?z)) -> (buy ?x)

The most specific pattern should be placed toward the front of the left-hand side of the rule

A specific pattern will generally have the smallest number of matching facts in the facts list and will have the largest number of variable bindings which constrain other patterns.

Control: Meta-Rules

- Meta-Rules

  Use Meta-Rules to divide rules into classes. Choose one class over another at a given point.

  This implements domain-dependent knowledge about which set of rules to use during reasoning.

  CLIPS provides a Module-construct with similar effects.

Categories of Rules

- Salience values are arbitrary; often what we want is that a certain class of rules are considered before others.
- This can be built into the rules themselves using a kind of 'tag'
- Another rule can be implemented to change status to the next group of rules.