Artificial Intelligence:
Search Methods for Problem Solving

Rule Based Expert Systems

A First Course in Artificial Intelligence: Chapter 6

Deepak Khemani
Department of Computer Science & Engineering
IIT Madras

Problem Decomposition

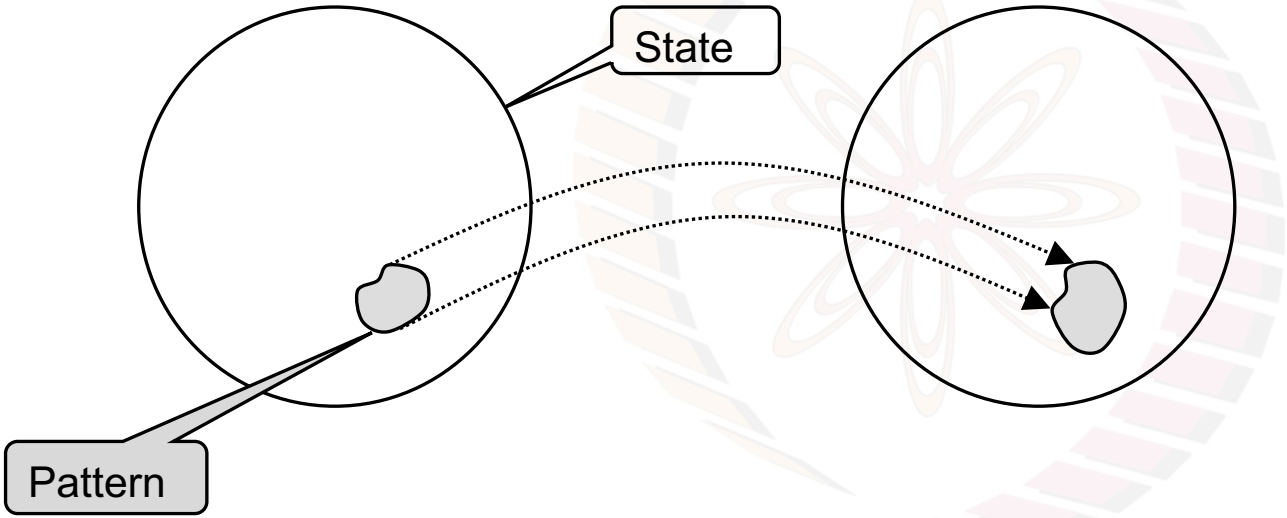
Recap

- So far our view of *problem solving* using search has been *state centered*
  - the *state space* is the arena for search
  - the *solution* is expressed as a *sequence of states*
  - even when we talk of solution space, the solution, for example for the TSP problem, is expressed in terms of states.
- Problem decomposition takes a goal directed view of problem solving
  - the emphasis is on breaking up a problem into smaller problems
    - like in backward state space planning: goal → subgoals
  - primitive problems are labeled SOLVED
  - … otherwise they are LIVE and have to be refined
    - like in the SSS* game playing algorithm

Patterns in the Domain

- Focus so far on
  - transition and search in the state space
  - transition and search in the solution space
- Problem decomposition
  - goals to subgoals
  - goal trees and AO* search
- Next - patterns in the state
  - what is inside the MoveGen function?
  - a collection of pattern–action responses
  - we make that explicit now
  - dismantle the MoveGen into individual moves
  - like in Planning algorithms

Pattern Directed Inference Systems



A *rule* looks at a part of a state which matches a pattern and modifies it to result in a new state

MoveGen: Underneath the Hood

State: A set of sentences in some language

Pattern: A subset of the sentences

Modifying the state in a piecewise fashion

Move: Pattern → Action

Declarative Programming

- A rule associates an *action* with a *pattern* in the state
  - also called a *production*
  - unifying format for heuristic knowledge, business rules, and actions
  - basis for a Turing complete programming language
- The user or programmer only states the rules
  - *very* popular in the business environment, *e.g.* lending by banks
  - the programmer *only* specifies the *pattern*→*action* association
  - she does not specify *when* an action should be executed
  - different from imperative programming
- A search strategy
  - explores which rules are applicable, and
  - which one to apply
  - user has a choice of strategies

Rule Based Production Systems

A working memory (WM)

- represents the current state
- contains a set of records or statements, known as working memory elements (WMEs)
- a model of the short term memory (STM) of the problem solver

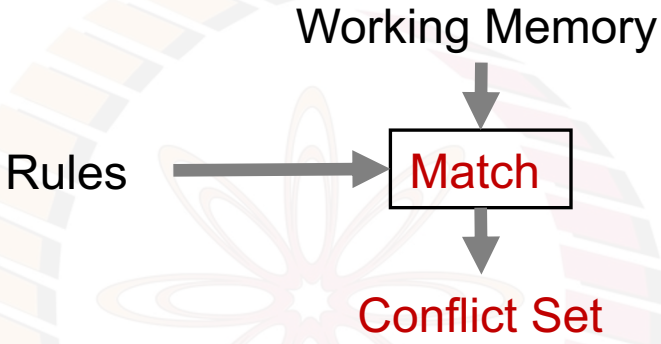
Each rule or a production

- represents a move
- has a name or an identifier
- has one or more pattern on the left hand side (LHS)
- has one or more action on the right hand side (RHS)
- a model of the long term memory (LTM) of the problem solver

An inference engine (IE)

- matches patterns in *all* rules with *all* WMEs
- picks *one* matching rule to fire or execute the actions in the rule
- repeats the process till some termination criterion

The Inference Engine: Match



Repeat
Match
Resolve
Execute

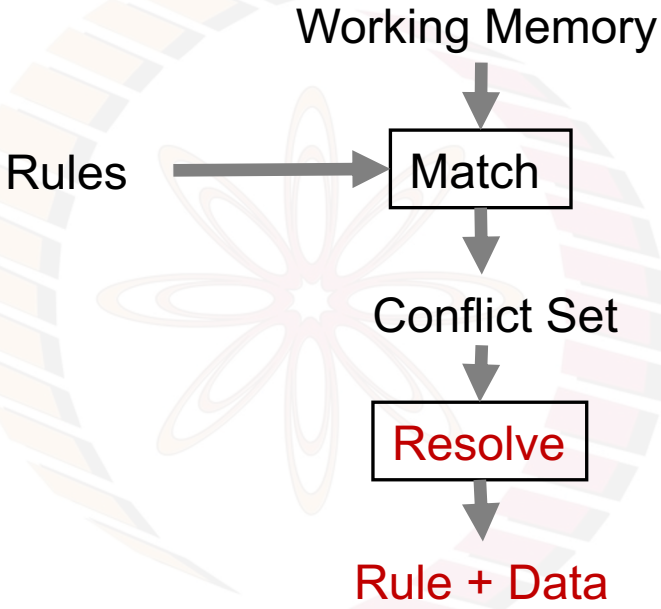
The Match algorithm takes the set of rules and the set of working memory elements and generates the *conflict set*.

Each element in the conflict set is a rule along with identities of matching WMEs.

The *conflict* to be resolved is *which* rule to select from the set of matching rules.

The Inference Engine: Resolve

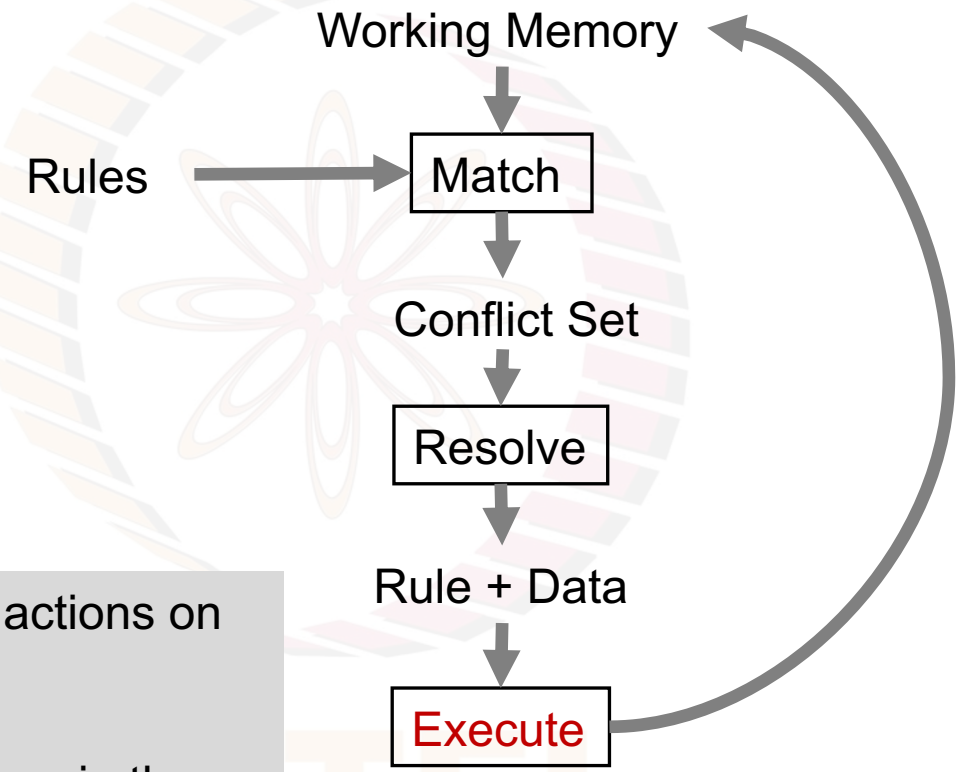
Repeat
Match
Resolve
Execute



Resolve selects one rule along with the matching working memory elements.

It encapsulates the strategy for search.

The Inference Engine: Execute



Repeat
Match
Resolve
Execute

Execute implements the actions on the RHS of a rule.

Actions may make changes in the working memory, which is updated

OPS5: a production system language

- Originated in CMU in the 1970s
  - Charles Forgy
- ”Official Production System” language
- The first rule based language
  - more than a search based problem solver
  - a declarative programming language
  - a Turing complete language
- Form: (p ruleName LHS → RHS)
  - p for production
- Expert Systems
  - programs with knowledge acquired from domain experts
  - knowledge in the form of rules
  - e.g. R1 for configuring Vax computer systems

OPS5 syntax: Working Memory

The data structure is a structured record

Class name with a set of attributes

Attribute names marked by ^

The data is a collection of attribute-value pairs bunched together

Order of attributes not important

Default value of attribute is NIL

For example,

```
(student ^name sudhir
         ^age 19
         ^semester 3
         ^discipline math
         ^degree MSc)
```

OPS5: Working Memory

Simon and Newell modeled the working memory (WM) as the *short term memory (STM)* of the problem solver. It contains the data the solver is currently working on.

The *working memory* is a collection of *working memory elements* (WMEs).
The WMEs are *indexed* by a *time stamp*,
indicating the order in which they were added to the WM.
For example,

Time stamp | Class name | Attribute | Value

1. (next ^rank 1)
2. (totalMarks ^student Sangeeta ^marks 97 ^rank nil)
3. (totalMarks ^student Suresh ^marks 63 ^rank nil)
4. (totalMarks ^student Akshai ^marks 92 ^rank nil)
5. (array ^index 1 ^value 29)
6. (array ^index 2 ^value 12

OPS5 syntax: Rules: LHS

The LHS of a rule is a *set of patterns*, that conform to the syntax of the WMEs.
The value field in patterns can have variables and Boolean tests.
There may even be *negative patterns*.

Ex: IF the next rank to be assigned is <r> note: <variable>
and there is a student with marks <m> and rank nil
and there is no student with higher marks and rank nil
THEN... do... whatever.

```
(p ranking
   (next ^rank <r>)
   (totalMarks ^student <s> ^marks <m> ^rank nil)
  −(totalMarks ^marks > <m> ^rank nil)
 →
   some RHS)
```

R1 or XCON

The R1 (internally called XCON, for eXpert CONfigurer) program was a production-rule-based system written in OPS5 by John P. McDermott of CMU in 1978 to assist in the ordering of DEC's VAX computer systems by automatically selecting the computer system components based on the customer's requirements.

ASSIGN-UB-MODULES-EXCEPT-THOSE-CONNECTING-TO-PANELS-4
IF: THE CURRENT CONTEXT IS ASSIGNING DEVICES TO UNIBUS MODULES
AND THERE IS AN UNASSIGNED DUAL PORT DISK DRIVE
AND THE TY PE OF CONTROLLER IT REQUIRES IS KNOWN
AND THERE ARE TWO SUCH CONTROLLERS NEITHER OF WHICH HAS ANY DEVICES ASSIGNED TO IT
AND THE NUMBER OF DEVICES THAT THESE CONTROLLERS CAN SUPPORT IS KNOWN
THEN:
ASSIGN THE DISK DRIVE TO EACH OF THE CONTROLLERS
AND NOTE THAT THE TWO CONTROLLERS HAVE BEEN ASSOCIATED
AND THAT EACH SUPPORTS ONE DEVICE

from this paper by John McDermott

OPS5 syntax: Conditions in the LHS

The *value* field in patterns can have *variables* and *Boolean tests*.

Equality
= Same type as and equal to
<> Not same type as or not equal to
<=> Same type as

Integer or Floating-point only
<, <=, >, >=

Conjunctions
{ > 3.0 < 3.5 }
> 3.0 AND < 3.5

Disjunctions
<< Monday Wednesday Friday >>
Monday OR Wednesday OR Friday

OPS5 syntax: Rules: RHS

Note:
All actions happen *concurrently*
Variable values from Match

The RHS of a rule is a set of actions.

- Make a new WME
- Remove an existing WME
- Modify = Remove + Make
- other actions like Read, Write, Load, Halt ...

Ex: (p ranking
(next ^rank <r>)
(totalMarks ^student <s> ^marks <m> ^rank nil)
-(totalMarks ^marks > <m> ^rank nil)
→
(Modify 1 ^rank (<r> + 1))
(Modify 2 ^rank <r>))

Modify the WME matching the1st pattern in the rule

IF you have found the record with the highest marks
THEN... assign rank <r> to the student, and increment <r>

OPS5 syntax: Rules: RHS

```
(p ranking
   (next ^rank <r>)
   (totalMarks ^student <s> ^marks <m> ^rank nil)
  -(totalMarks ^marks > <m> ^rank nil)
→
    (Modify 1 ^rank (<r> + 1))
    (Modify 2 ^rank <r>) )
```

Identifier for WME

Alternatively
one can use variables

```
(p ranking
   { <rank> (next ^rank <r>)}
   { <total> (totalMarks ^student <s> ^marks <m> ^rank nil)}
  -(totalMarks ^marks > <m> ^rank nil)
→
    (Modify <rank>  ^rank (<r> + 1))
    (Modify <total> ^rank <r>) )
```

Sorting

An array is a set of WMEs of class “array”
with attributes “index” and “value”

The objective is to sort the values in increasing order.

The following rule *spots* two values out of place, and swaps them

```
(p swapSort
   (array ^index <i> ^value <X>)
   (array ^index {<j> > <i>} ^value {<Y> < <X>})
→
   (modify 1 ^value <Y>)
   (modify 2 ^value <X>) )
```

Repeated firing of the rule will eventually sort the values.

Match

A rule has a match in the WM if

a) Each positive pattern has a matching WME. Unsigned patterns are positive by default.
b) There is no WME in the WM that matches a negative pattern

```
p rule
   pattern 1
   pattern 2
   .
   .
→
   actions...
```

A pattern in a rule matches a WME in the WM if

a) the class name of the pattern = class name of the WME
b) each attribute condition in the pattern matches the attribute value in the WME
c) attributes not mentioned in the pattern but present in the WME are ignored

Attribute Matching: Conditions in the LHS

Attribute in rule	Matching value in WME	
constant	identical constant	
<variable>	any constant	this makes rules universal
Boolean		
=	same type as and equal to	<n>=3, <x>=<y>
<>	not same type as or not equal to	
<=>	same type as	
Boolean on Numbers		
<, <=, >, >=	numbers satisfying constraint	<n>< 3, <x> >= <y>
Conjunction		
{test1 ... testN}	must match *all* tests	
Disjunction		
<<test1 ... testN}>>	must match *some* test	

Conflict Set

```
(p swapSort
      (array ^index <i> ^value <X>)
      (array ^index {<j> > <i>} ^value {<Y> < <X>})
→
      (modify 1 ^value <Y>)
      (modify 2 ^value <X>) )

(p ranking
      (next ^rank <r>)
      (totalMarks ^student <s> ^marks <m> ^rank nil)
     -(totalMarks ^marks > <m> ^rank nil)
   →
      (Modify 1 ^rank (<r> + 1)
      (Modify 2 ^rank <r>)
```

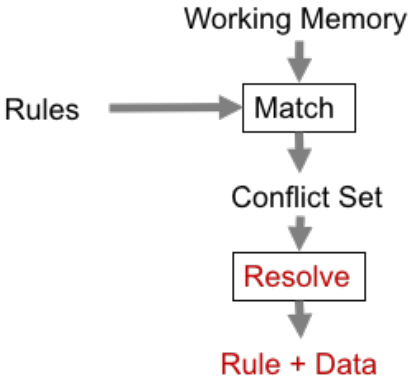
Working Memory

1. (array ^index 1 ^value 7)
2. (array ^index 2 ^value 9)
3. (array ^index 3 ^value 3)
4. (array ^index 4 ^value 8)
5. (next ^rank 2)
6. (totalMarks ^student Rashmi ^marks 89 ^rank 1)
7. (totalMarks ^student Eva ^marks 79 ^rank nil)
8. (totalMarks ^student Anil ^marks 79 ^rank nil)
9. (totalMarks ^student Salil ^marks 69 ^rank nil)

Conflict Set

{<swapSort 1 3>, <swapSort 2 3>, <swapSort 2 4>, <ranking 5 7>, <ranking 5 8>}

Conflict Resolution



The following rule *spots* two values out of place...

```
(p swapSort
   (array ^index <i> ^value <X>)
   (array ^index {<j> > <i>} ^value {<Y> < <X>})
→ .....
```

Given a large set of WMEs, the rule may match many pairs of elements!

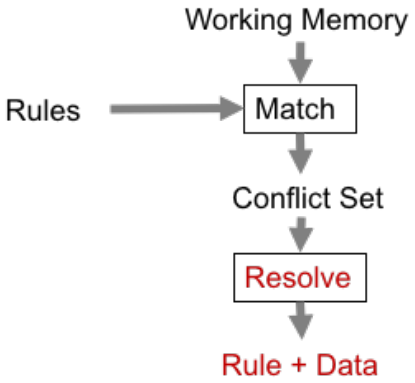
The set of matching rules with data is called the conflict set (CS)

The Inference Engine selects one rule from the conflict set

- point to ponder – can we execute many rules in parallel?

Conflict resolution: choosing *which* rule to execute or fire next

Conflict Resolution Strategies



The key to effective problem solving is to make informed choices!

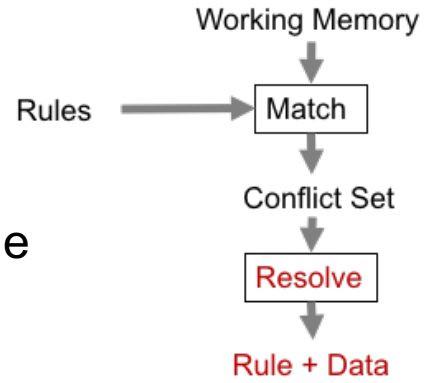
In heuristic search this was done by the heuristic function h(N).

In rule based systems, the rules encapsulate such knowledge gleaned from human experts.

When more than one rule matches the data the inference engine has to make the choice of which rule instance to execute
- somewhat analogous to which candidate to inspect during search

We examine several Conflict Resolution Strategies

Conflict Resolution: Refractoriness



This says that a rule instance may fire only once with a set of matching WMEs. This is particularly relevant when the selected rule does not modify the WMEs matching it's preconditions.

- else only this rule would keep firing

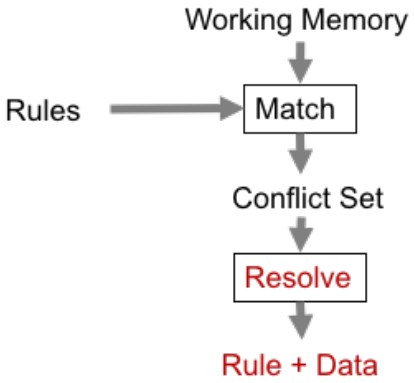
The idea of refractoriness comes from the way neurons fire in animal brains. When a neuron has received excitation that crosses its threshold, it fires once, and then waits for new signals to come in.

"the insensitivity to further immediate stimulation that develops in irritable and especially nervous tissue as a result of intense or prolonged stimulation"

- Merriam Webster Dictionary

Refractoriness happens naturally in the Rete Algorithm

Conflict Resolution: Lexical Order



This says that of all the rules that have matching instances
choose the first one that the user has stated.

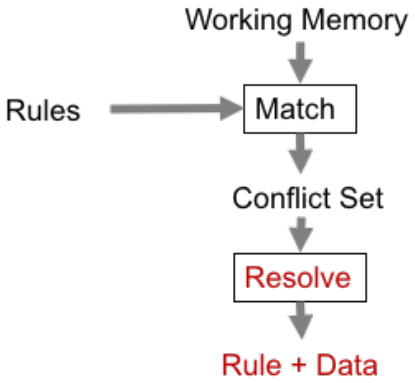
And if a rule has multiple instances with different data,
then choose the instance that matches the earlier data.

This strategy places the onus of this choice on the user.
The user is more like a programmer.

This strategy is used in the programming language Prolog.

Prolog thus deviates from the idea of declarative programming envisaged by pure logic programming, in which the user would only state the relation between input and output.

Conflict Resolution: Specificity



This says that of all the rules that have matching instances
choose the instance of the rule that is *most specific*.

Specificity can be measured
in terms of the *number of tests* that patterns in rules need.

The intuition is that the more specific the conditions of a rule are,
the more appropriate the rule is likely to be in the given situation

- remember that the Working Memory
  models the Short Term Memory of the problem solver
- rules constitute the problem solvers knowledge
  and reside in the quasistatic Long Term Memory

Specificity can facilitate default reasoning....

Default Rules: An example from bridge bidding

```
(p pass-rule
   (turn-to-bid ^player <x>)
   (next ^after <x> ^is <y>)
→
   (make (bid ^player <x> ^bidName pass)
   (modify 1 ^player <y>) )
```

Default action: Pass
(unless you have something to communicate)

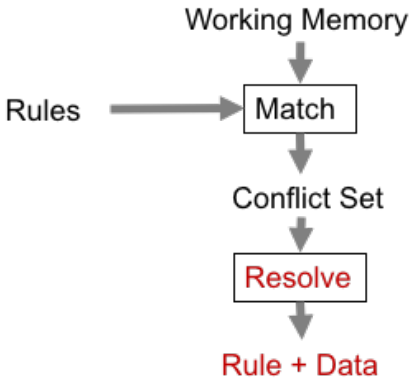
Specificity selects the more specific rule when both rules match

```
(p 1N-opening
   (turn-to-bid ^player <x>)
   (next ^after <x> ^is <y>)
   (hand ^player <x> ^points <<15 16 17>> ^shape balanced)
  −(bid ^player {<y> <> <x> ^round 1)
→
    (make (bid ^player <x> ^round 1 ^type opening
                                    ^bidName notrump ^denomination 1)
    (modify 1 ^player <y>))
```

Holding 15-17 high card points with balanced shape, bid 1NT if no one else has bid anything

Exercise: Write rules to conclude that birds fly but not if they are penguins

Conflict Resolution: Recency



This says that of all the rules that have matching instances
choose the instance that has the most recent WME.

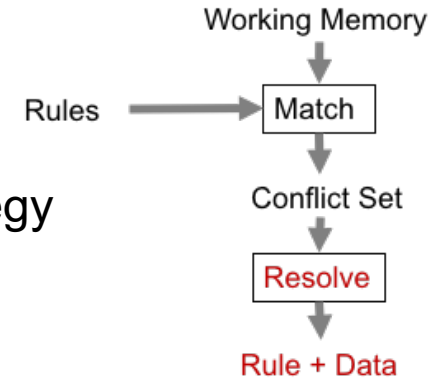
Recency can be implemented
by looking at the time stamps of the matching WMEs.

The intuition is that when a problem solver adds a new element to the Working Memory
than any rule that matches that WME should get priority.

Recency can facilitate ”a chain of thought” in reasoning.

The Conflict Set can be maintained as a priority queue.

Conflict Resolution: MEA



The implementation of the language OPS5 from CMU has a strategy called MEA (Means Ends Analysis) based on the problem solving strategy espoused by Simon and Newell.

The idea is to partition the set of rules based on the *context* and focus on one partition at a time. One can think of each partition as solving a specific subgoal or reducing a specific difference.

The context is set by the *first pattern* in a rule. All rules in the same partition have the same first pattern.

The MEA strategy applies *Recency* to the first pattern in each rule, and *Specificity* for the remaining patterns.

Towards an efficient implementation of a forward chaining rule based system

The Rete Network

The origin of the name *rete* is based on the English word, *riet* in middle English, itself derived from the Latin word *rete* meaning network, for "*An anatomical mesh or network, as of veins, arteries, or nerves*." -- www.yourdictionary.com/rete

Definition of *rete*

1 : a network especially of blood vessels or nerves : plexus

2 : an anatomical part resembling or including a network

https://www.merriam-webster.com/dictionary/rete

Business Rule Management Systems

Charles L. Forgy developed the Rete algorithm in the late 70's but it took a little while for the algorithm to become widely popular, when Business Rules technology finally emerged.

- Most BRMS vendors developed their own algorithm that are known in the elite circles as XRete or Uni-Rete..
- Rete 2 is the version that Forgy developed which broke the legendary "Rete wall" which referred to the dramatic performance degradation when the number of objects in working memory increased.
- Rete 3 is the evolution of Rete 2 developed for RulesPower and subsequently integrated with Blaze Advisor.

From this blog (2011) by Carole-Ann Berlioz of Sparkling Logic

Rete-NT (a trade secret)

- The new OPSJ Rete-NT engine from Production Systems Technology uses the latest Rete incarnation from Charles Forgy.
- The Rete-NT engine works with any Rete-based BRMS, including those from IBM, Oracle, Fair Isaac, Red Hat, and Pegasystems.
- It is available to the vendors (licensed at $5,000 per CPU) or can be used directly by PST clients with the OPSJ rule syntax.
- The new generation of his algorithm
  called RETE-NT is much faster than Rete3.
- How did he do it? Well, he would not say …

Trade secret…!

From this blog (2011) by Carole-Ann Berlioz of Sparkling Logic

Faster Processing with the Rete Net

The best usage of the Rete network I have seen in a business environment was likely Alarm Correlation and Monitoring.

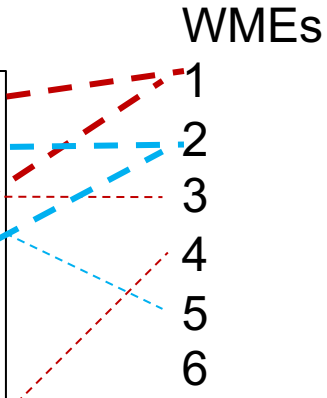
- ... a “stateful” kind of execution where alarms are received over time as they occur.
- ... the actual Telecom network is made of thousands of pieces of equipment to keep in “mind” while processing the alarms, there is no wonder that Rete outperforms brute force.
- When one alarm is raised on one Router, the Rete network does not have to reprocess all past events to realize that we reached the threshold of 5 major alerts on the same piece of equipment and trigger the proper treatment, eventually providing a probable diagnostic.
- Hours of processing time turned into seconds in Network Management Systems. Fabulous.

From this blog (2011) by Carole-Ann Berlioz of Sparkling Logic

Intra-cycle savings

```
(p grade-AI-C
      (student ^name <n> ^rollNo <r> ^age <a> ^year <y>)
      (course ^student <r> ^subject AI ^marks {<m> >59  <= 70} ^grade nil)
  →
       (Modify 2 ^grade C) )

(p grade-ML-C
      (student ^name <n> ^rollNo <r> ^age <a> ^year <y>)
      (course ^student <r> ^subject ML ^marks {<m> >49  <= 60} ^grade nil)
  →
       (Modify 2 ^grade C) )
```



CS = { ..., <grade-AI-C,1,3>,
<grade-AI-C,2,5>,
<grade-ML-C,1,4>,
...}

1. (student ^name Sneha ^rollNo 201835 ^age 20 ^year 3) ... in 5 courses
2. (student ^name Sunil ^rollNo 201706 ^age 21 ^year 4) ... in 3 courses
3. (course ^student 201835 ^subject AI ^marks 66 ^grade nil)
4. (course ^student 201835 ^subject ML ^marks 50 ^grade nil)
5. (course ^student 201706 ^subject AI ^marks 62 ^grade nil)
6. (course ^student 201706 ^subject ML ^marks 69 ^grade nil)

In practice only one grading rule

```
(p grade-assignment
      (cutoffs ^subject <s>  ^low <low> ^high <h> ^grade <g>)
      (student ^name <n> ^rollNo <r> ^age <a> ^year <y>)
      (course ^student <r> ^subject <s> ^marks {> <low>  <= <h>} ^grade nil)
  →
       (Modify 3 ^grade <g>) )
```

1. (student ^name Sneha ^rollNo 201835 ^age 20 ^year 3)
2. (student ^name Sunil ^rollNo 201706 ^age 21 ^year 4)
3. •
4. •
5. (course ^student 301835 ^subject AI ^marks 66 ^grade nil)
6. (course ^student 301835 ^subject ML ^marks 50 ^grade nil)
7. (course ^student 201706 ^subject AI ^marks 62 ^grade nil)
8. (course ^student 201706 ^subject ML ^marks 69 ^grade nil)
9. •
10. •
11. (cutoffs ^subject AI ^low 59 ^high 70 ^grade C)
12. •
13. •

Inter cycle savings

Let us say say there are 300 students

who between them have taken 1200 courses

Initially there are 1200

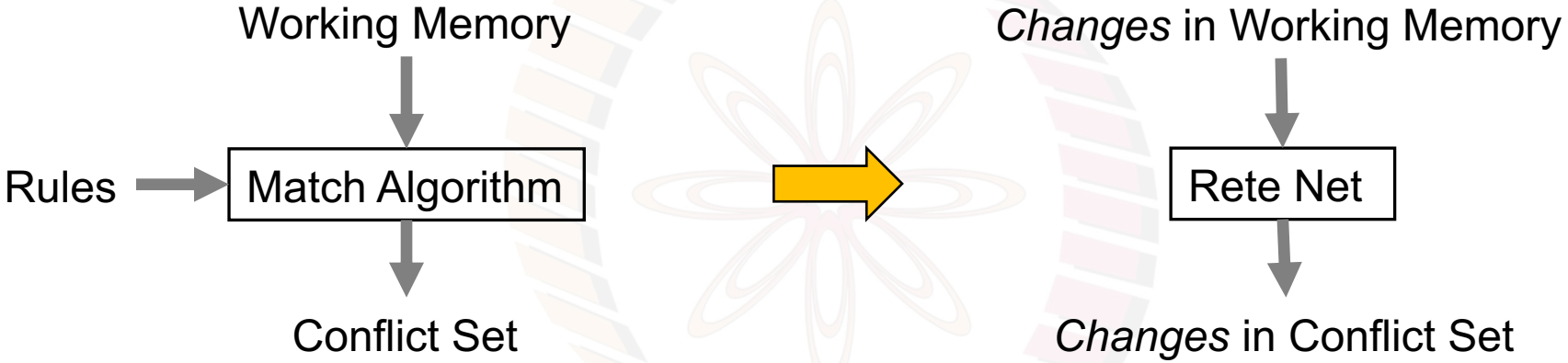
instances of grade-assignment rule in the conflict set

One of these instances is selected, and fires

The remaining 1199 are still matching

why compute their match again?!

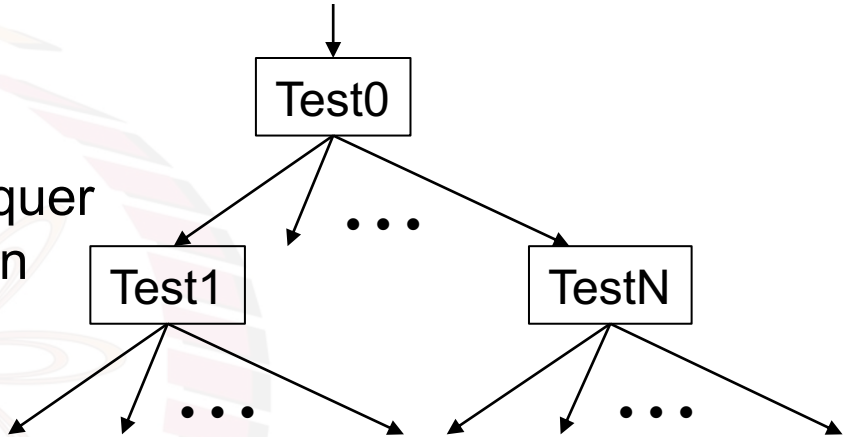
No more brute force Match



Discrimination Trees

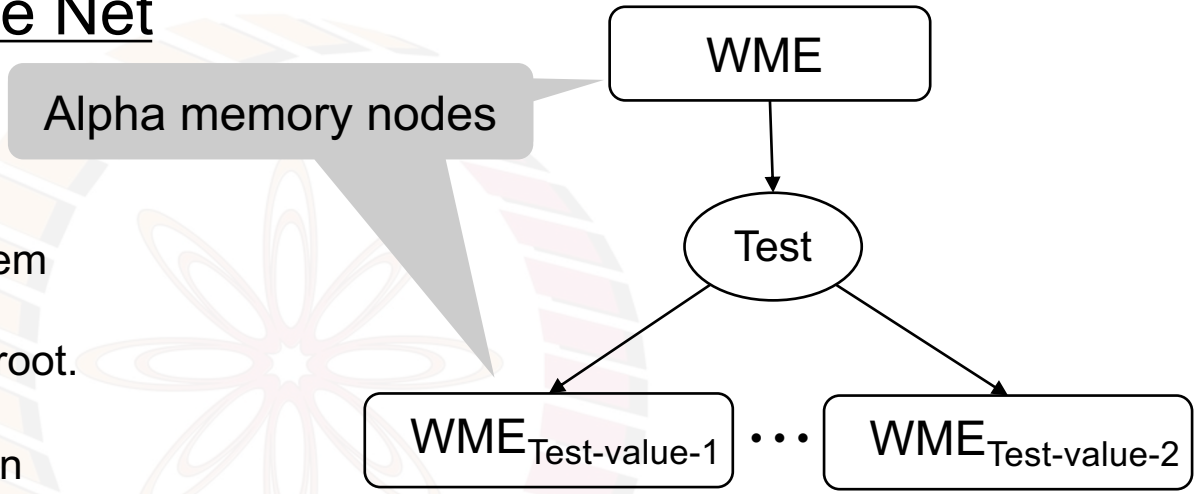
A popular approach to work with large amount of data is to use the Divide & Conquer approach and route the query or data token via a sequence of tests...

- Binary Search Trees
- B-trees
- Decision Trees
- KD-trees
- Quadtrees
- Tries



The top half of a Rete Net is a discrimination tree

Alpha nodes in the Rete Net



Alpha nodes serve as
the WM of the Rule Based System

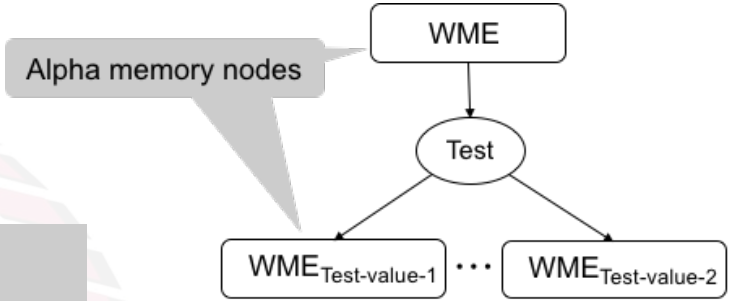
A WME-token is inserted at the root.
<+ WME> for an ADD action
<− WME> for a DELETE action

They travel down the discrimination tree

The first test, usually, looks at the Class-name and separates the tokens.
Subsequent tests look at the value of some attribute.

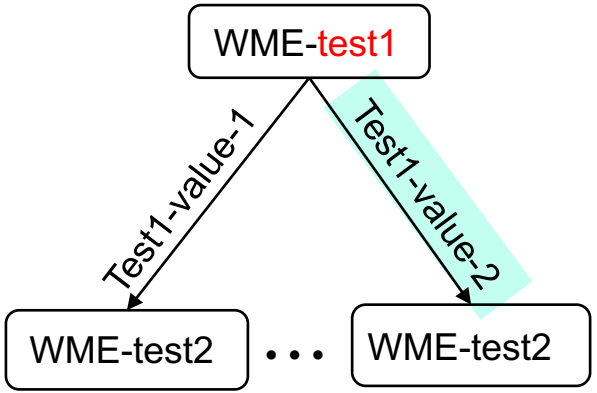
The key is to sequence the tests in such a manner
so that the common tests in different patterns
are higher up in the network, and shared between patterns.

Alpha nodes – compact diagrams



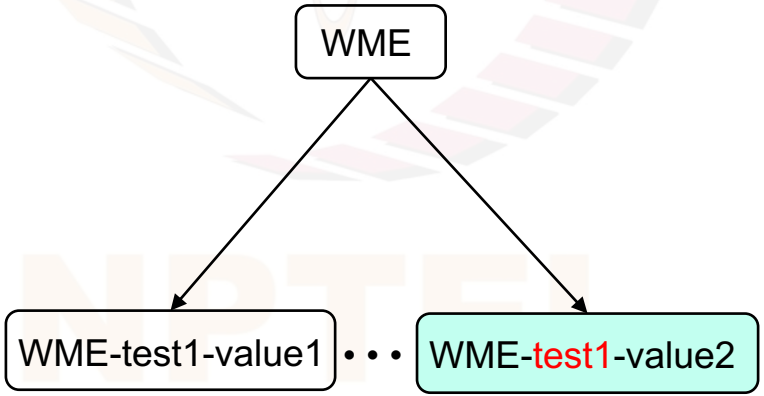
Test at current node
Edges mark test value

Closer to BST, Decision Tree

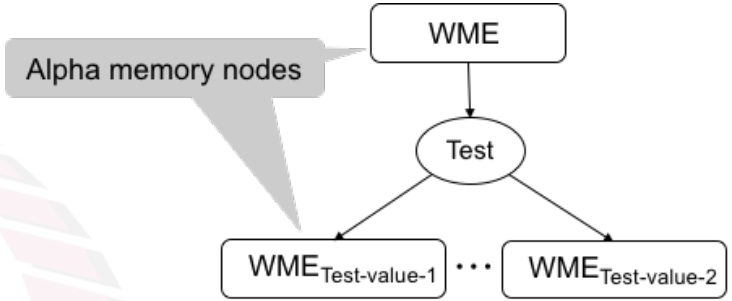


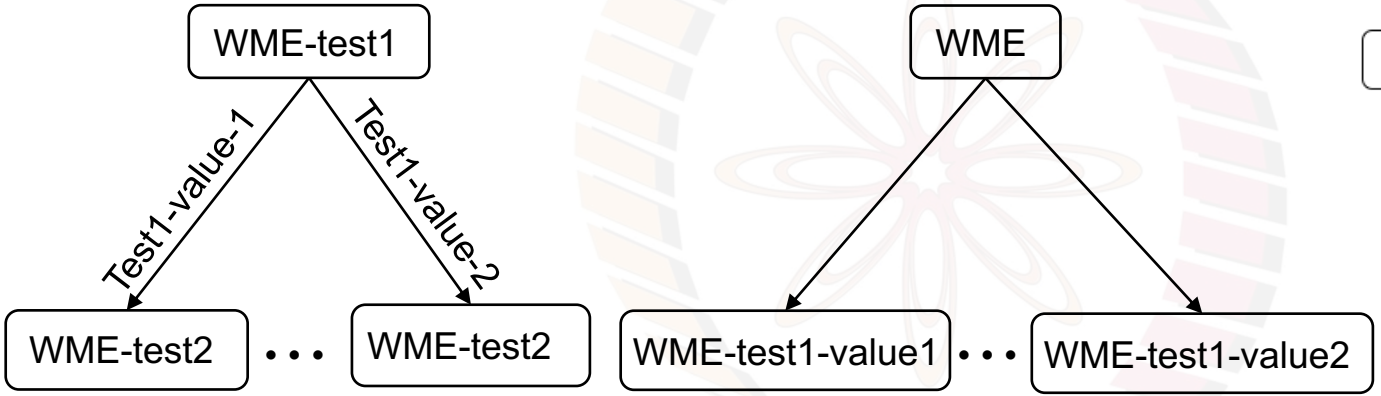
Test AND value at destination node

Identifies final location of WME



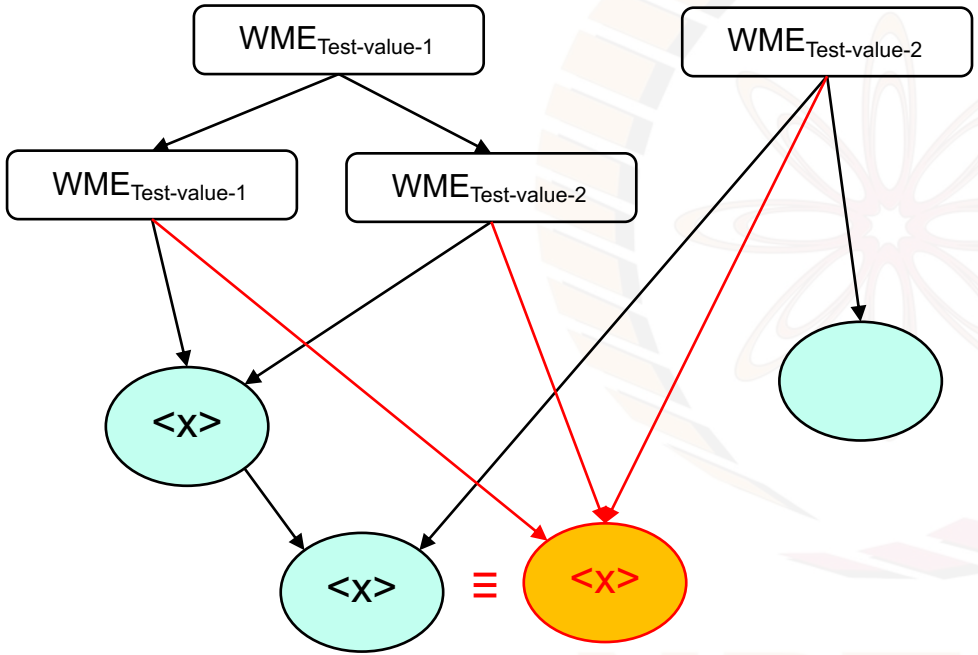
Alpha nodes – where WMEs dwell





IF a WME passes a test
THEN it moves to an appropriate alpha node at the next level
ELSE it gets stuck on the parent node

Beta nodes in the Rete Net



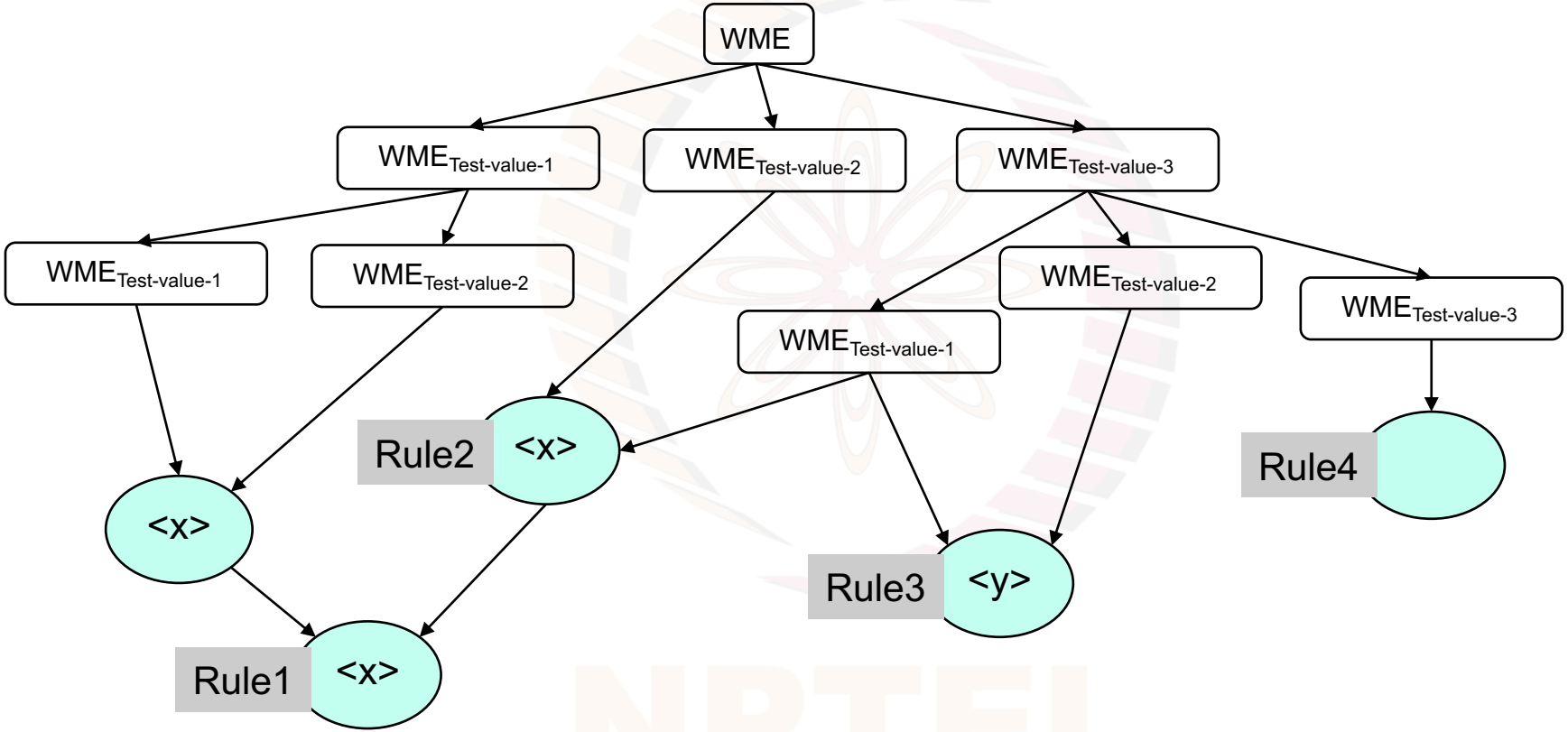
A join on 3 tokens

Beta nodes have at least one parent, more often more.

Two or more parents must satisfy a join condition – shared variable must have the same value

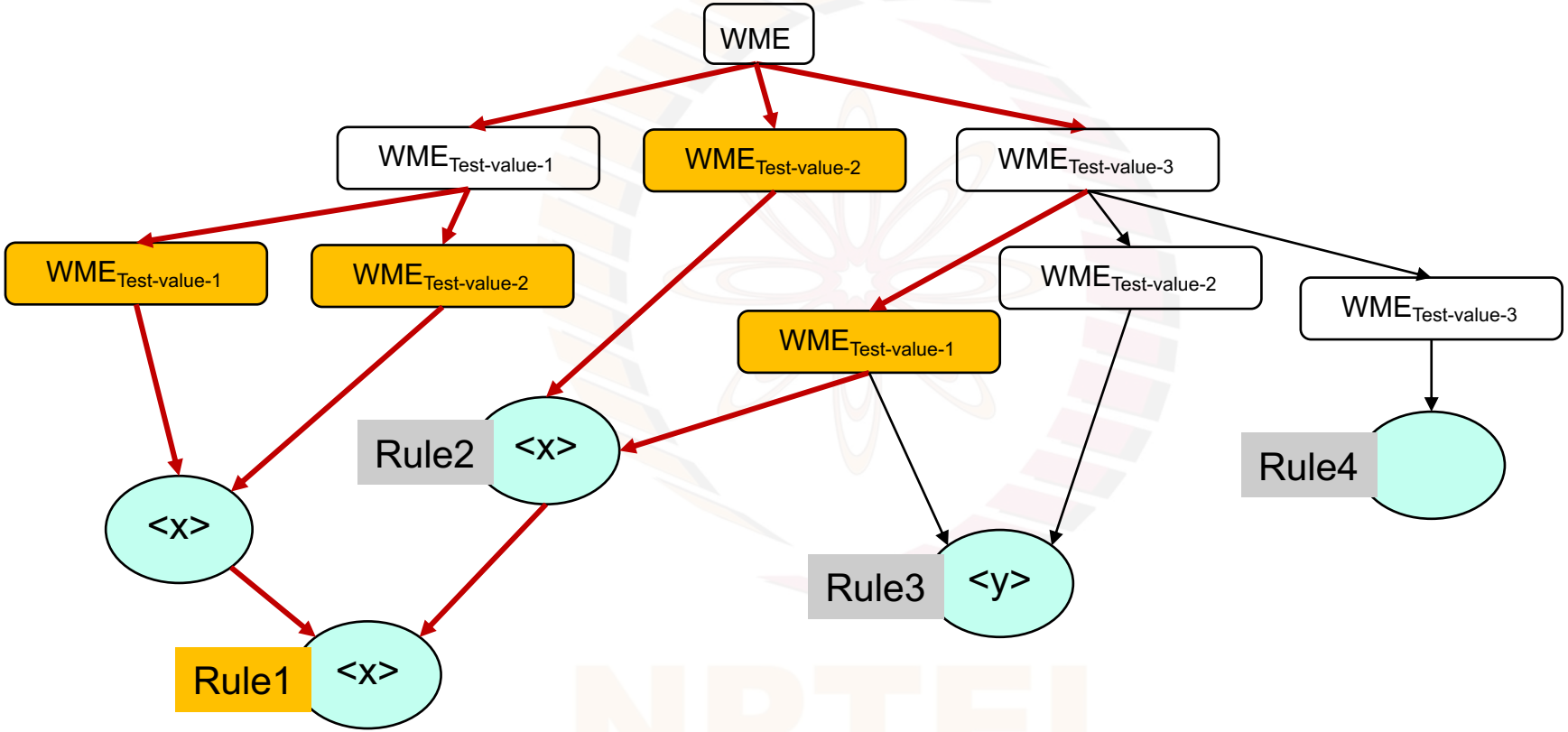
A rule instance can be attached to any beta node

Rule instances in the Rete Net



Observe that Rule2 is default version of Rule1

Rule instances in the Rete Net



Rule1 has four patterns, Rule 2 has only two

Pyramids and Cylinders

```
(p greenPyramid
   (block ^name <x>)
   (base ^block <x> ^shape square ^area >1)
   (side ^block <x> ^angle <90 ^surface plane ^color green)
   (top ^block <x> ^surface point)
→
   (make (class ^block <x> ^type greenPyramid)) )
```

If X is a block
whose base is a square with area greater than 1
whose side is green inclined plane surface
with a pointed top
Then
Add a WME saying X is a green pyramid

```
(p cylinder
   (block ^name <x>)
   (base ^block <x> ^shape circle ^area >1)
   (side ^block <x> ^angle 90 ^surface curved)
   (top ^block <x> ^surface flat)
→
   (make (class ^block <x> ^type cylinder)) )
```

If X is a block
whose base is a circle with area greater than 1
whose side is vertical curved surface
whose top is a flat surface
Then
Add a WME saying X is a cylinder

Wands and Domes

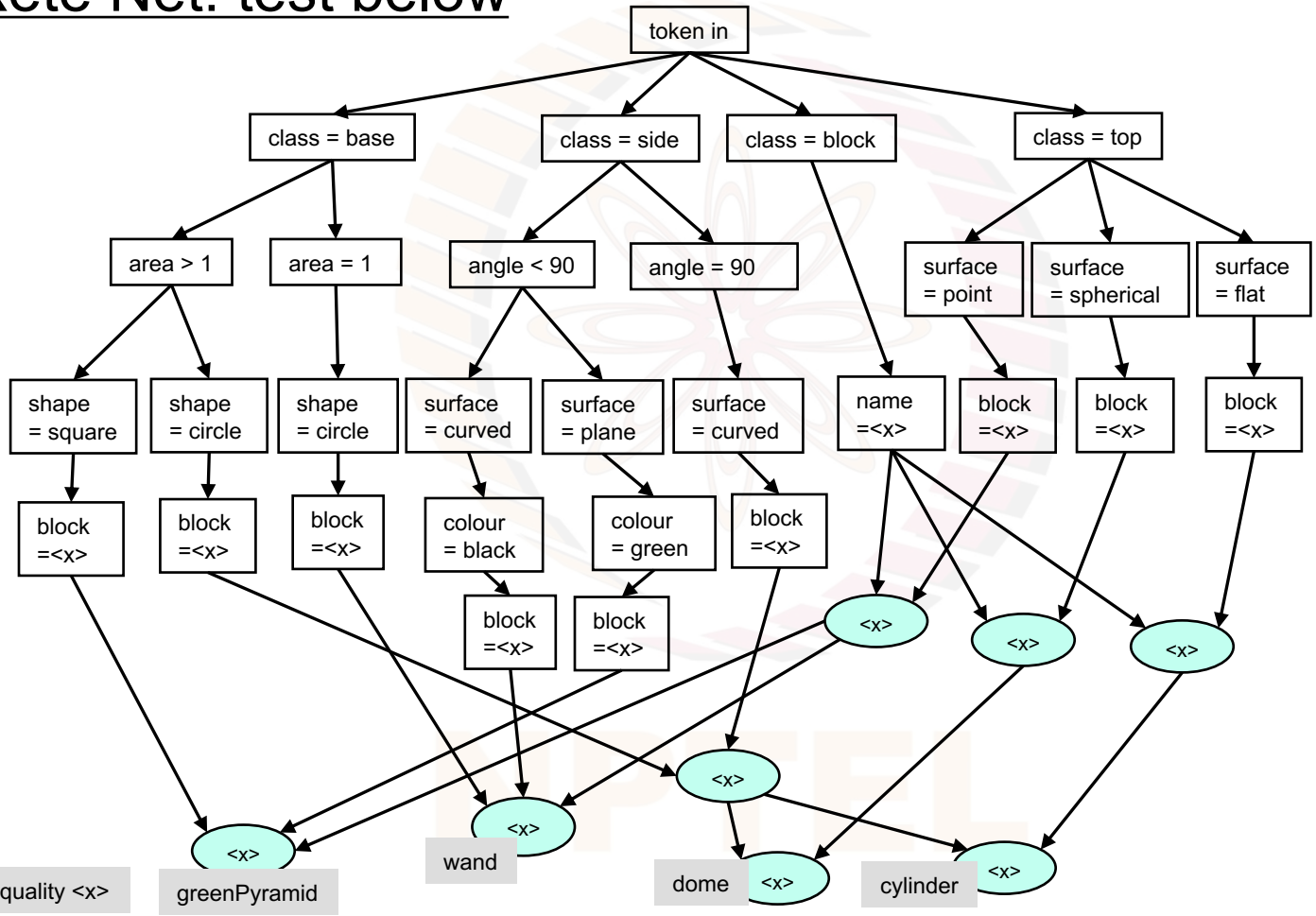
```
(p wand
   (block ^name    <x>)
   (base ^block <x> ^shape circle ^area 1)
   (side ^block <x> ^angle <90 ^surface curved ^color black)
   (top ^block <x> ^surface point)
→
   (make (class ^block <x> ^type wand)) )
```

circular small base
curved black sides
pointed top

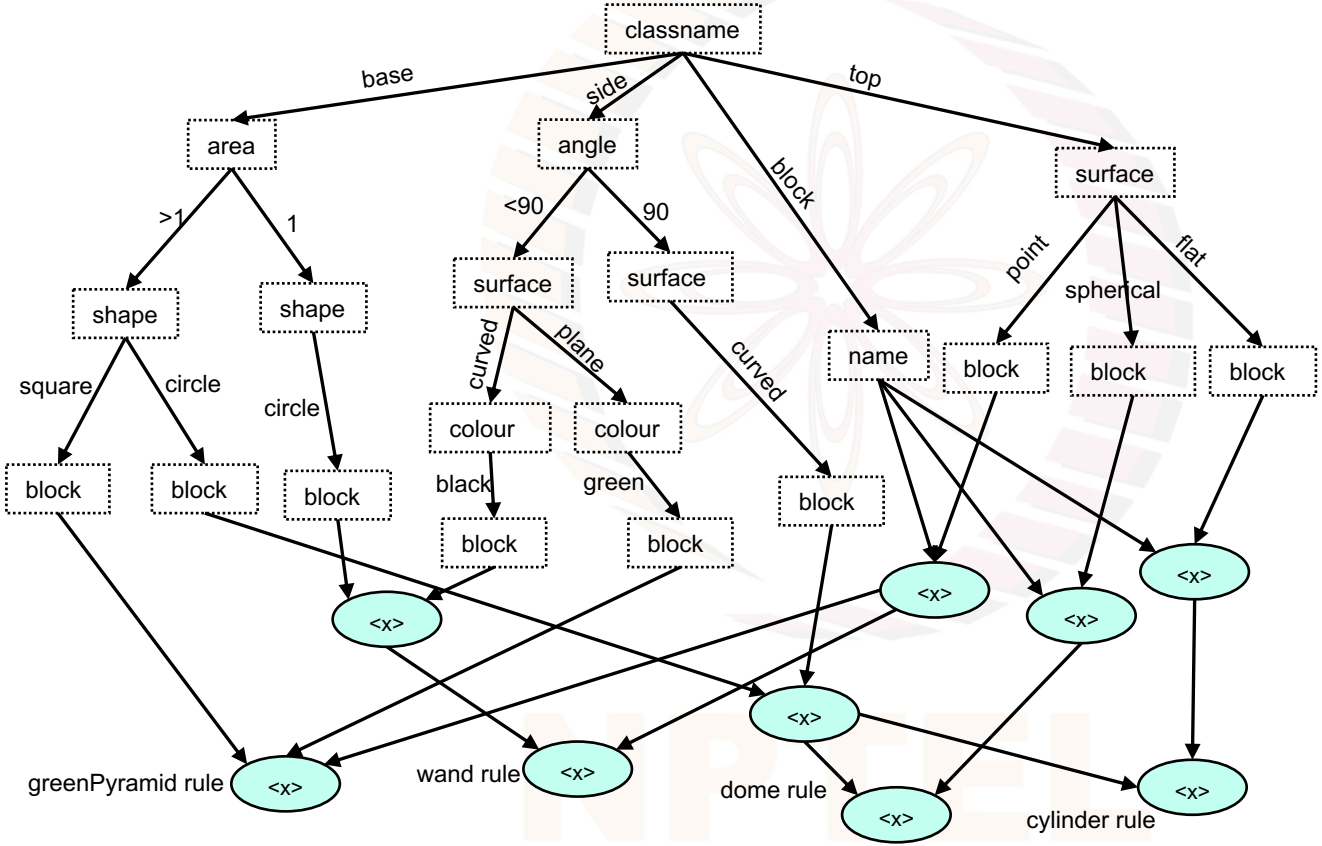
```
(p dome
   (block ^name    <x>)
   (base ^block <x> ^shape circle ^area >1)
   (side ^block <x> ^angle 90 ^surface curved)
   (top ^block <x> ^surface spherical)
→
   (make (class ^block <x> ^type dome) )
```

large circular base
curved vertical surface
spherical top

The Rete Net: test below



The Rete Net: test above

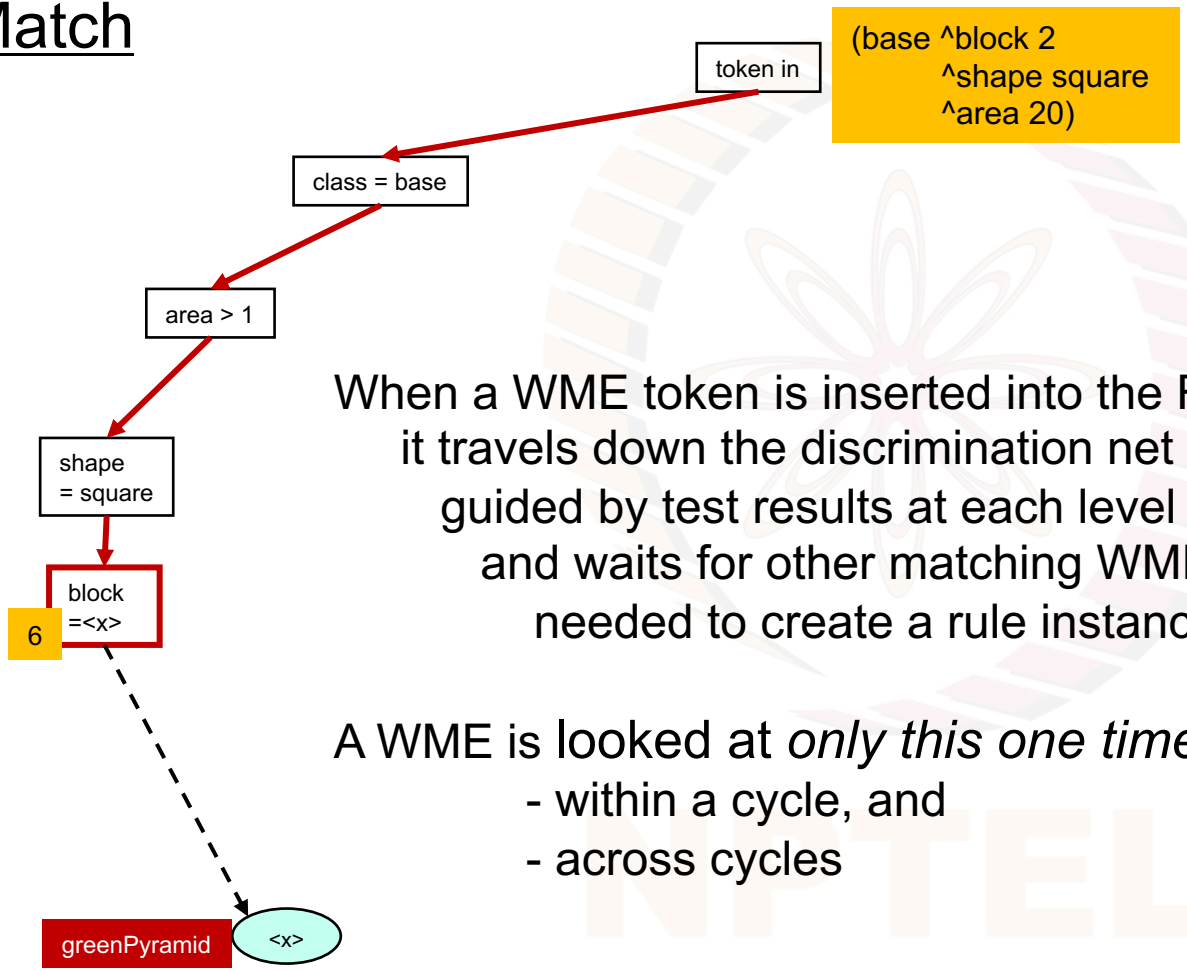


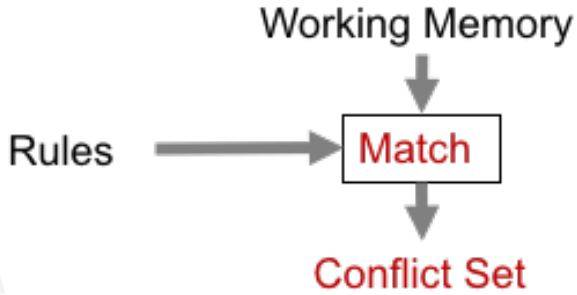
All Joins on equality of variable <x>

The Working Memory Elements

1. (block ^name 1 ^owner Harry)
2. (block ^name 2 ^owner Hermione)
3. (block ^name 3 ^owner Ron)
4. (block ^name 4 ^owner Draco)
5. (base ^block 1 ^shape circle ^area 1)
6. (base ^block 2 ^shape square ^area 20)
7. (base ^block 3 ^shape circle ^area 10)
8. (base ^block 4 ^area 15)
9. (side ^block 1 ^angle 80 ^surface curved ^color black)
10. (side ^block 2 ^angle 60 ^surface plane ^color green)
11. (side ^block 3 ^angle 90 ^surface curved)
12. (side ^block 4 ^angle 50 ^surface curved)
13. (top ^block 1 ^surface point)
14. (top ^block 2 ^surface point)

Match





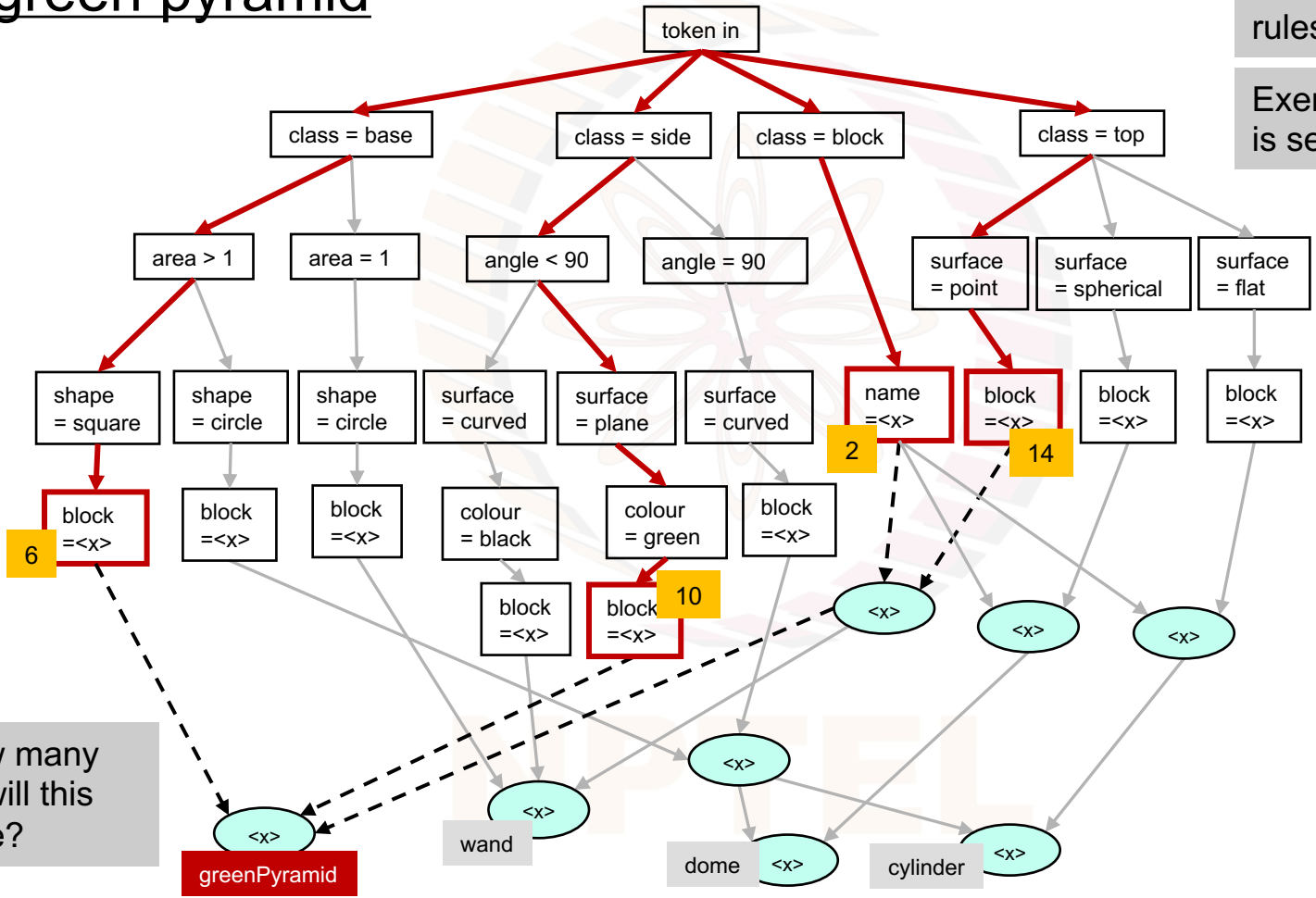
When a WME token is inserted into the Rete Net
it travels down the discrimination net
guided by test results at each level
and waits for other matching WMEs
needed to create a rule instance

A WME is looked at *only this one time*
- within a cycle, and
- across cycles

The green pyramid

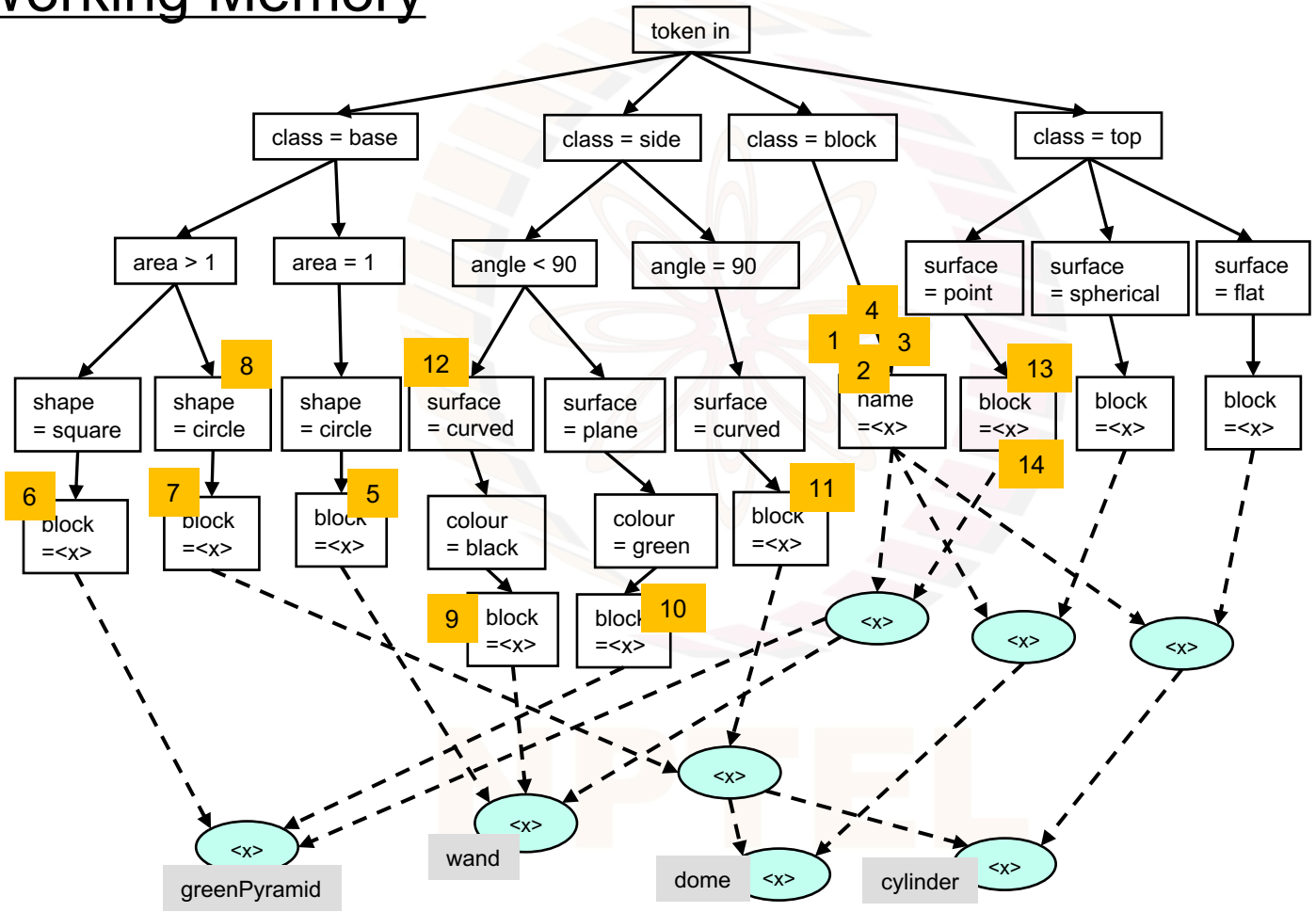
Exercise: Which other rules(s) are matching?

Exercise: Which rule is selected?



Q: How many times will this rule fire?

The Working Memory



Match → Resolve

When a positive token is dropped in, it travels down, and *may* trigger some rules whose other conditions have already been met.

- if it does, put these rule instances in a bucket -- equal *recency*

For every rule instance in the conflict set, the sum the lengths of the paths defines specificity.

- for *specificity* maintain a priority queue on the sum of lengths

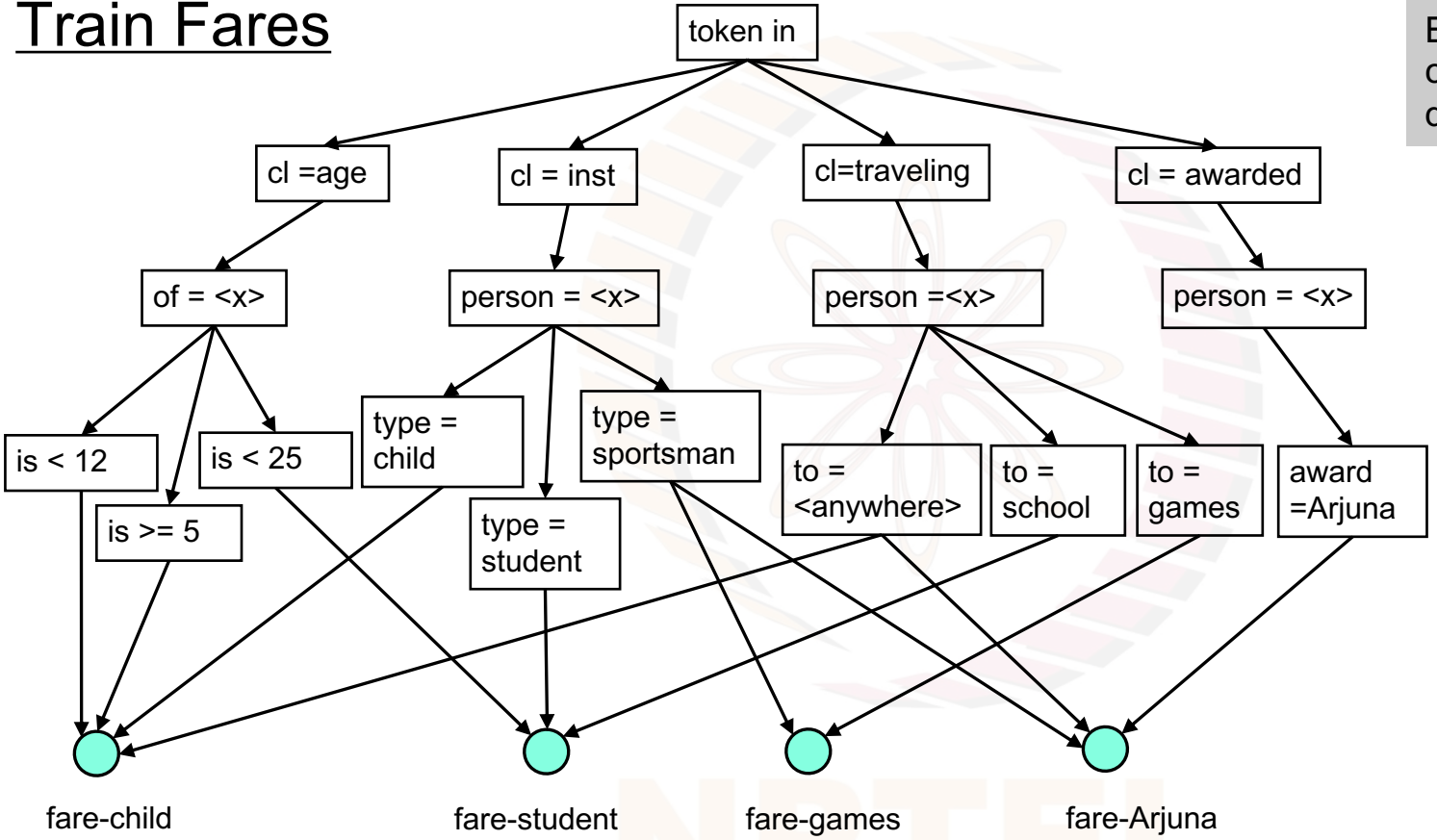
Once a rule instance is selected to fire, it is removed from the conflict set

- *refractoriness* is automatic

When a *negative token* (delete WME action) is dropped in,
it may go and exorcise some rules matching earlier

Rules with *negative patterns* need special treatment

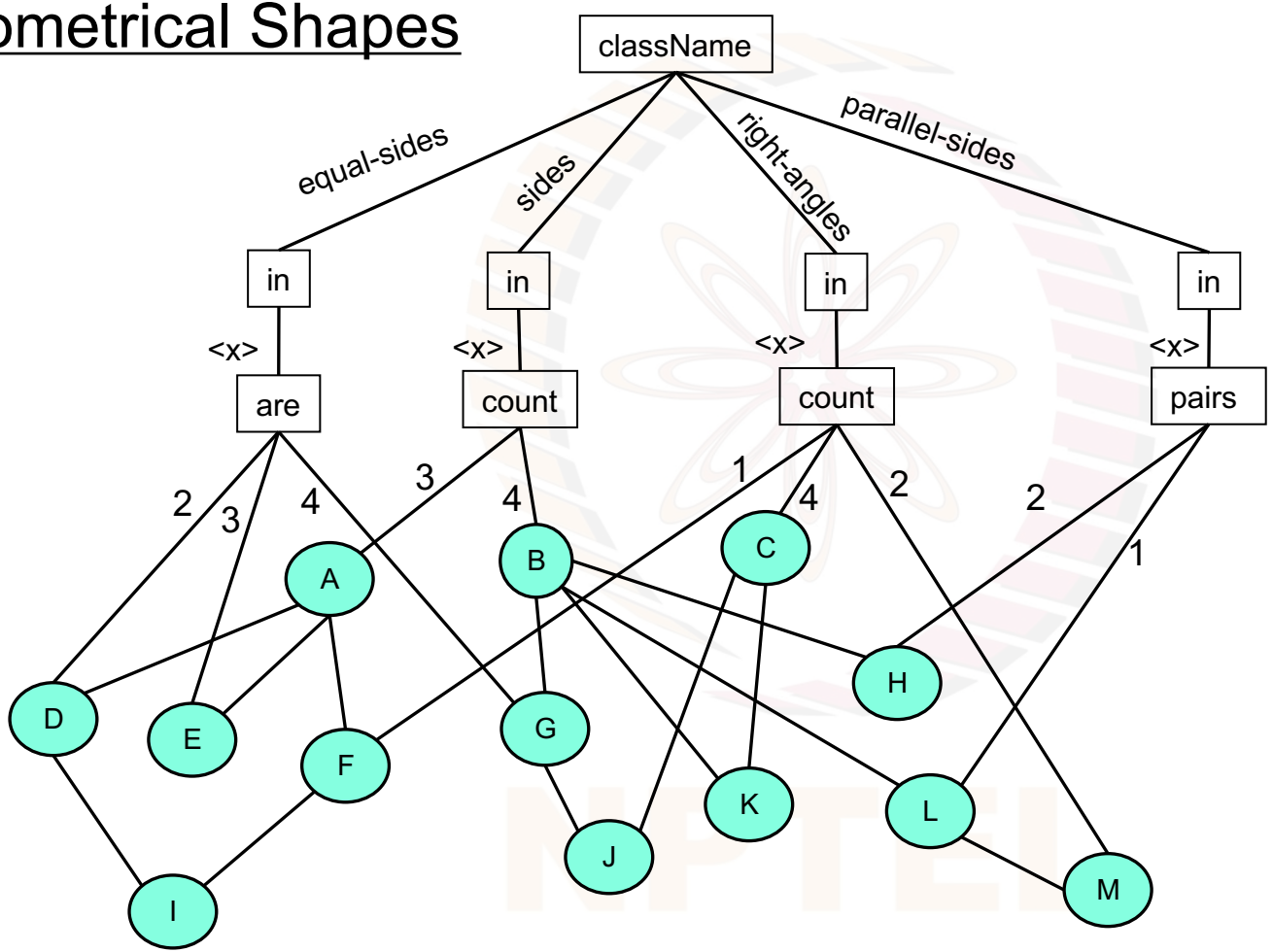
Train Fares



Exercise: Write the corresponding rules to determine fare category

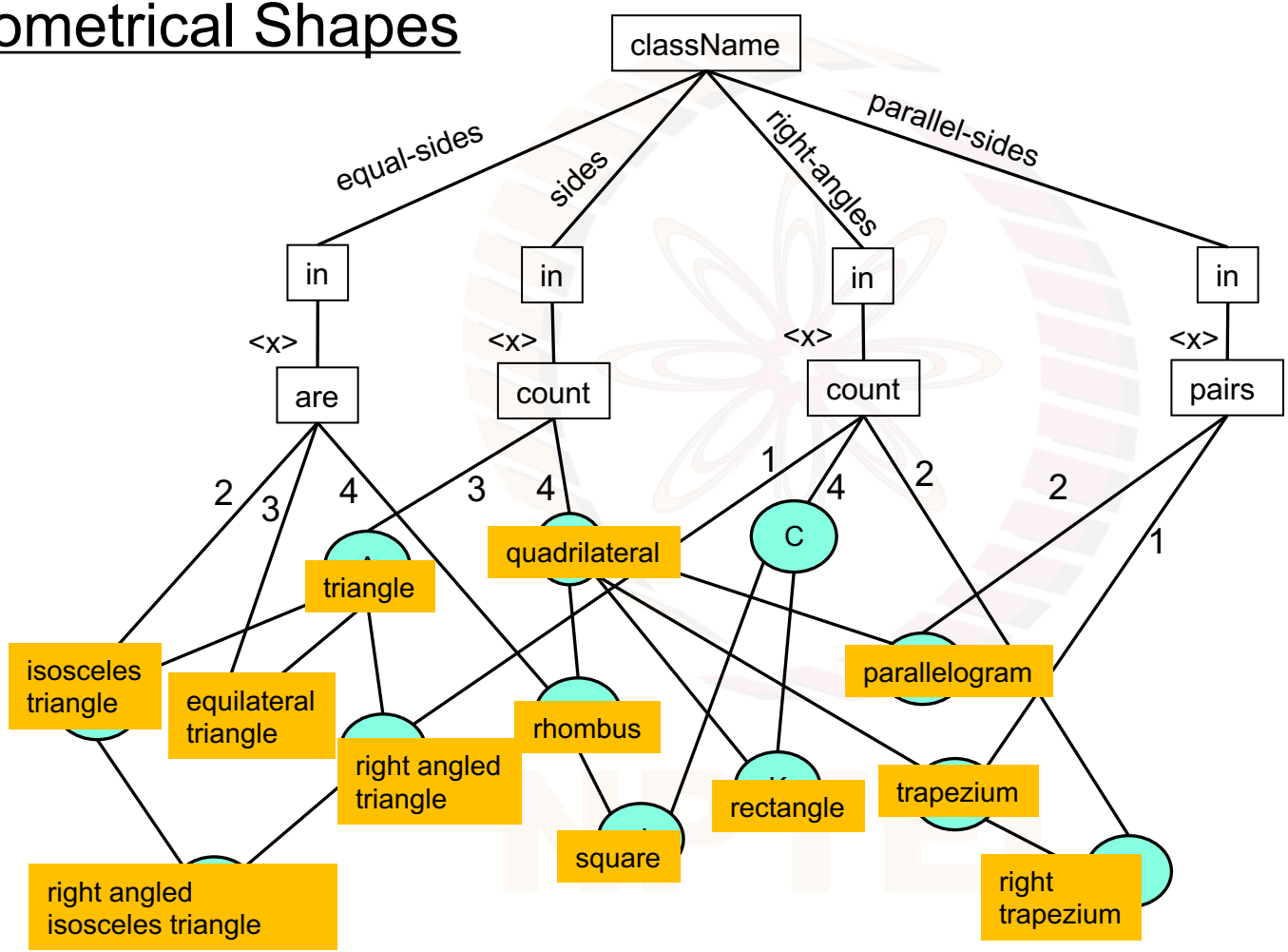
All Joins on equality of variable <x>

Geometrical Shapes



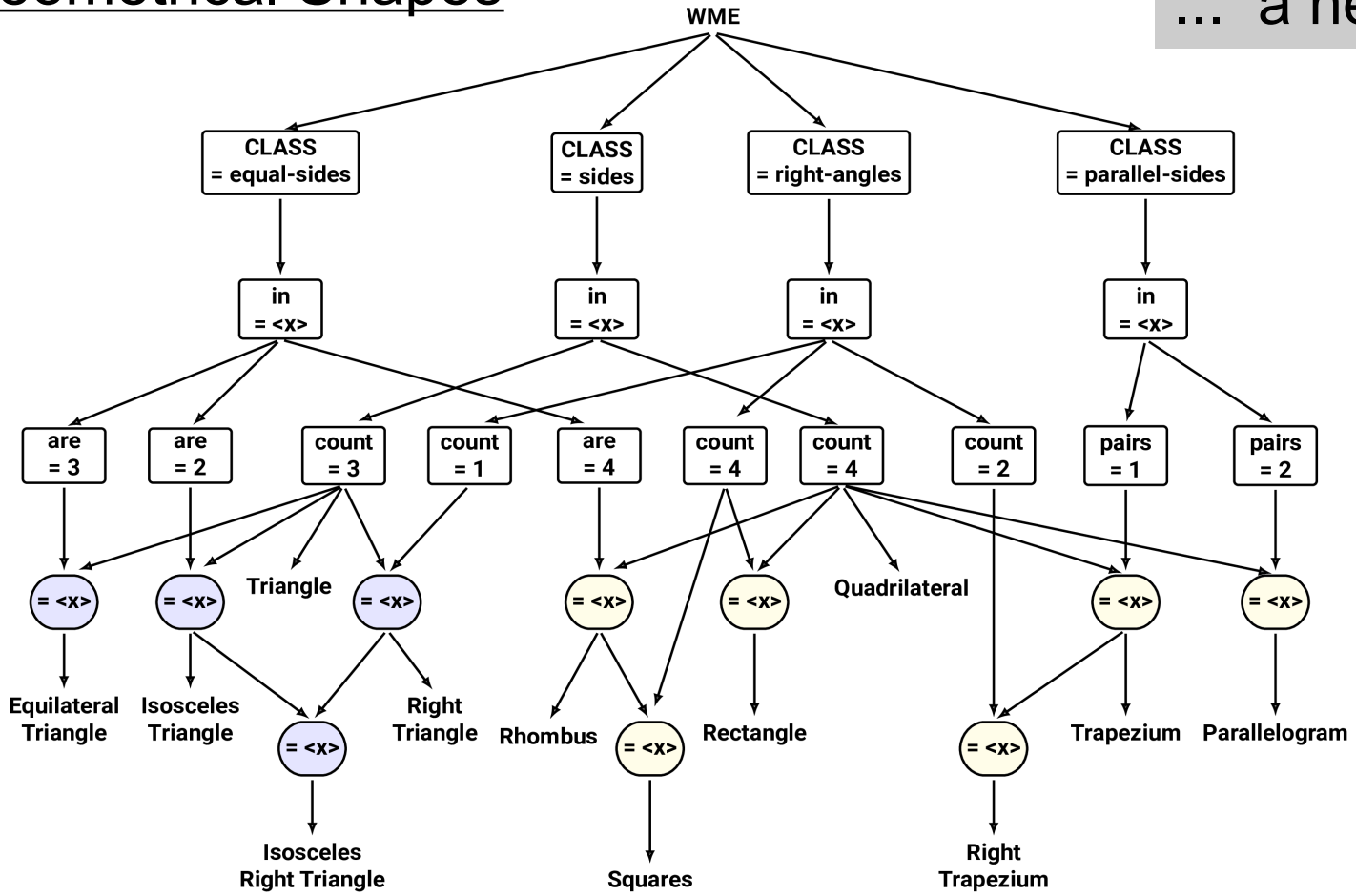
A. Triangle
B. Quadrilateral
C. ?
D. Isosceles Triangle
E. Equilateral Triangle
F. Right Triangle
G. Rhombus
H. Parallelogram
I. Right Isosceles Triangle
J. Square
K. Rectangle
L. Trapezium
M. Right Trapezium

Geometrical Shapes



Geometrical Shapes

... a neater diagram



A note on Inheritance

Consider the two rules for a rhombus and a square

A figure with 4 sides and all 4 sides being equal is a rhombus

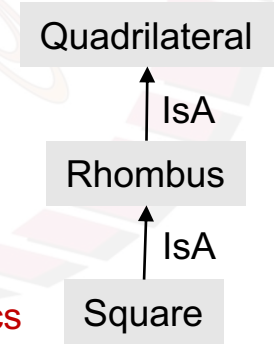
A figure with 4 sides and all 4 sides being equal and all 4 angles being right angles is a square

The square is more specific than the rhombus, which in turn is more specific than a quadrilateral

Default rules often have an underlying inheritance hierarchy

A square is a rhombus

A rhombus is a quadrilateral



Such relations are also studied in logic

If X is a square then X is a rhombus

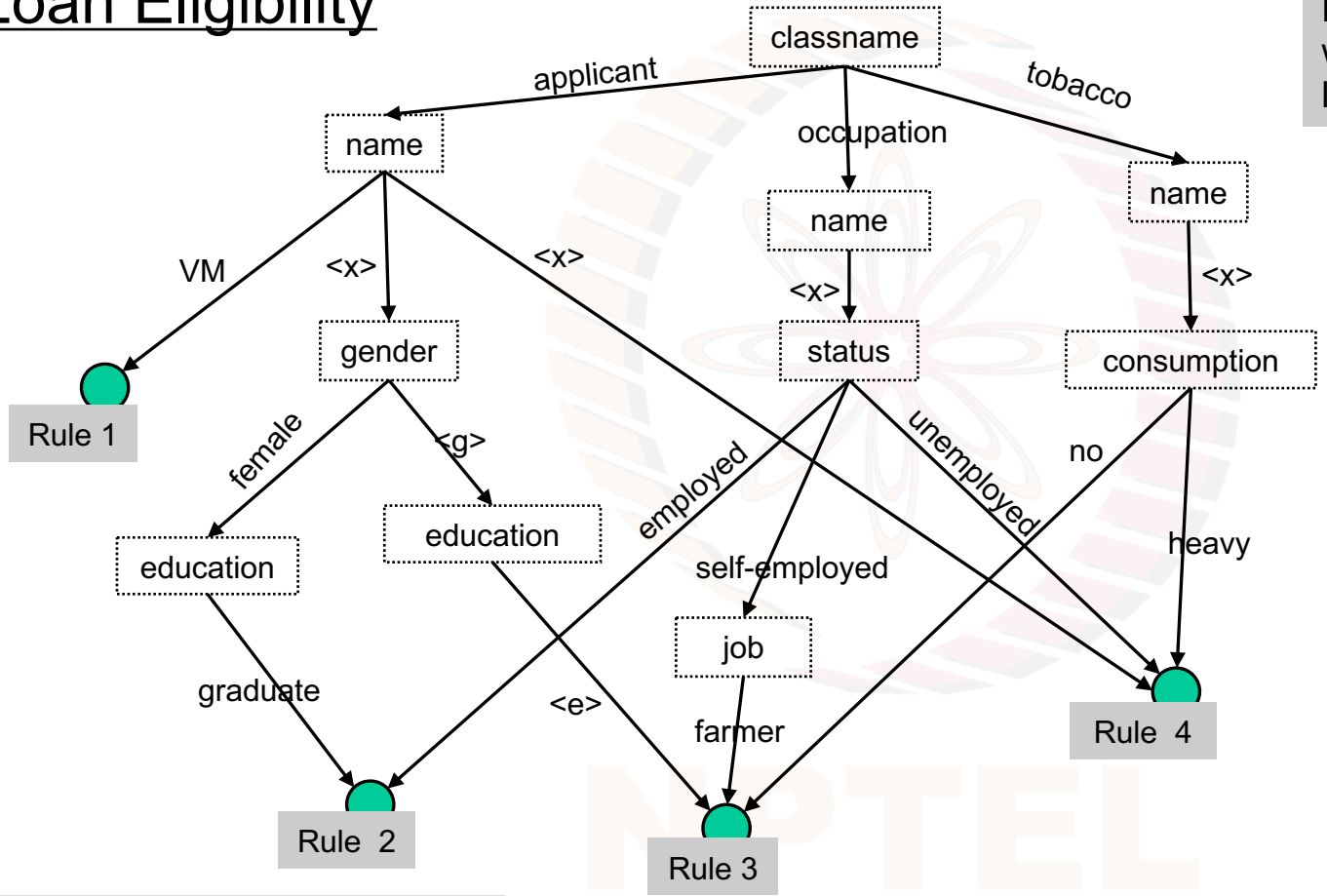
If X is a square then X is a quadrilateral

In particular they are concisely expressed in Description Logics

Discussed in detail in AI: Knowledge Representation & Reasoning

Loan Eligibility

Exercise: Assign rule names, write the rules, and fill in the likely RHS

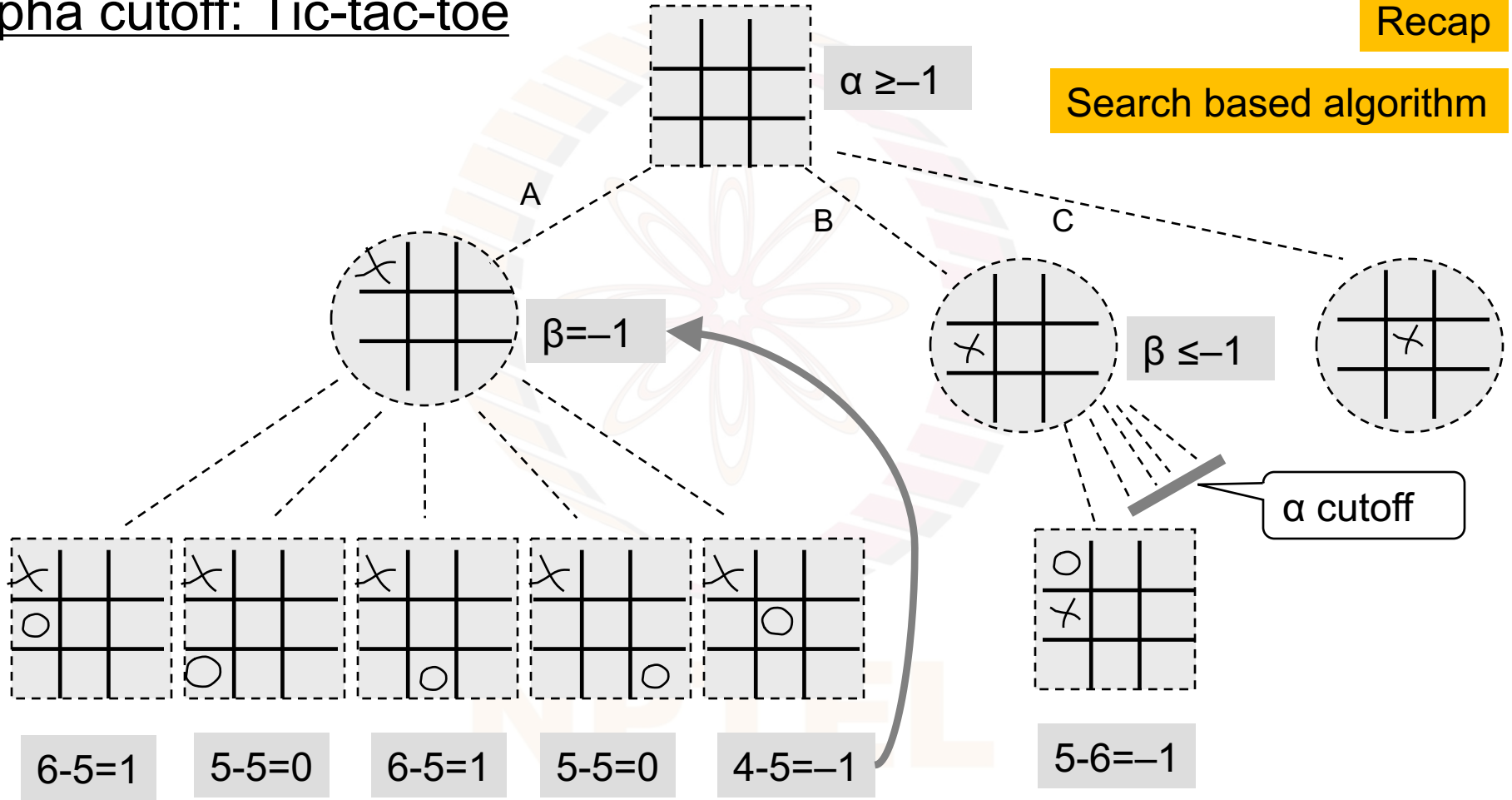


All Joins on equality of variable <x>

Alpha cutoff: Tic-tac-toe

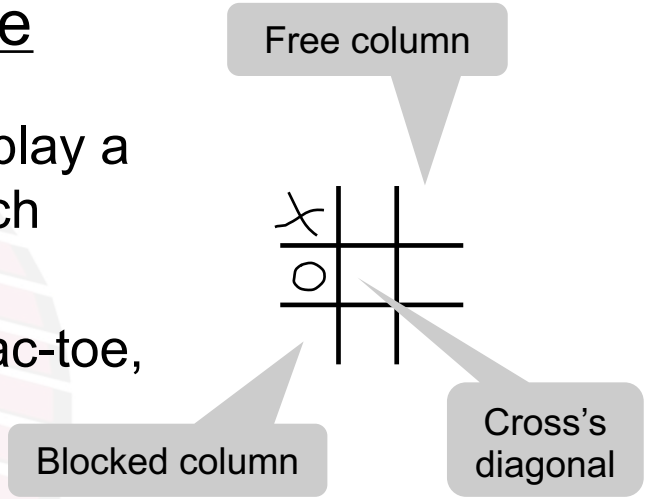
Recap

Search based algorithm



A Knowledge Based Approach to Tic-tac-toe

- Very often humans use acquired knowledge to play a game rather than use systemic lookahead search
  - for example in Chess opening game, or the end game
- Most humans have acquired heuristics for Tic-tac-toe, for example
  - corner squares are good
  - control of centre is good
  - create a fork
  - block opponents row, column or diagonal



Exercise: Write a set of OPS5 like rules to play Tic-tac-toe

How would you ensure that the conflict resolution strategy picks the rule to make the best move at any time?

What do you know?

An autonomous agent

operating in a dynamic world

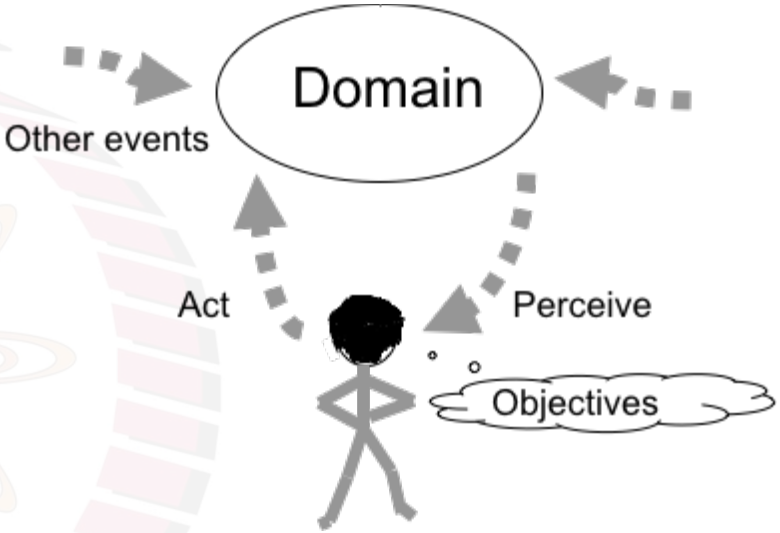
creates a representation of the world

senses the world around it

makes plans to achieve its goals

monitors the execution of those plans

and constantly updates what it knows



What else do you know?

An autonomous agent

operating in a dynamic world

...needs to make inferences about its world

Coming up next,

Representation and Reasoning in Logic

Another string in the bow of a problem solving agent

End : Rete Net