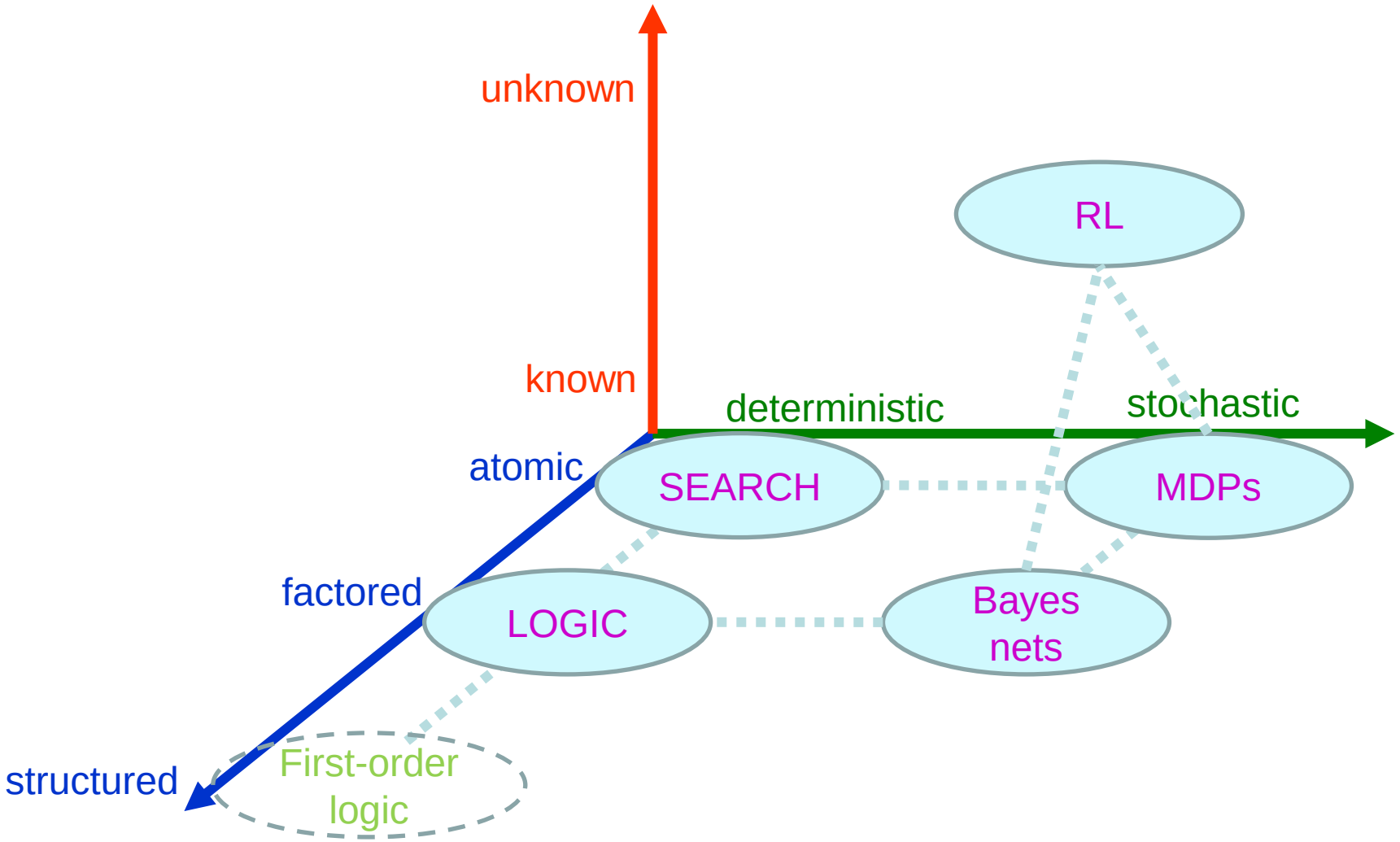
Artificial Intelligence

Production Base System



Outline

- Introduction
- Components
- Architecture
- Inference
  - Forward Chaining
  - Backward Chaining
- Problems with the Recognize-Act Cycle
  - Rete Matching Algorithm

Production Based System

- Oldest techniques of knowledge representation
- System consists:
  - **Knowledge base**: represented by production rules
  - **Working memory**: hold the matching patterns of data that causes the rules to fire
  - **Interpreter** (inference engine): decides which rule to fire, when more than one of them are concurrently firable
- Rule firing causes:
  - New consequences are added to the working memory
  - Old and unnecessary consequences of previously fired rules are dropped out from the working memory

Production Rule

- Production Rule:
  - PR1: Mother(m, c) ⇒ Loves(m, c)
  - Rule Maps: a *mother-child* relationship between (m, c) to a *love* relationship between the same pair
  - Rule implicates: *if "m" is a mother of child "c" then "m" loves "c"*

- Production Rule Structure:
  - PR1: $P_1(x) \wedge P_2(y) \wedge .. P_n(x, z) \Rightarrow Q_1(y) \vee Q_2(z) \vee .. Q_m(y, x)$

    Antecedent / Condition

    consequent / conclusion

    need not be always predicates

    object-attribute-value

  - P_i and Q_i are predicantes
  - x, y, z are variables
  - $\wedge$ and $\vee$ are logical AND, OR
  - $\Rightarrow$ if-then operators

Example

- PR2 : *if* (person age above-21) *&*
  (person wife nil) *&*
  (person gender male)
  then (person eligible for marriage)

  More General Rule

- PR3: *if* (Ram age 25) *&*
  (Ram wife nil) *&*
  (Ram gender male)
  then (Ram eligible for marriage).

  More Specific Rule

The Working Memory (WM)

- WM holds data either in the form
  - Clauses : Mother(*Rajmata_Jijabai*, *Chha. Shivaji*)
  - Object-attribute-value (OAV) : *Rajmata_Jijabai Spouse Shahaji*

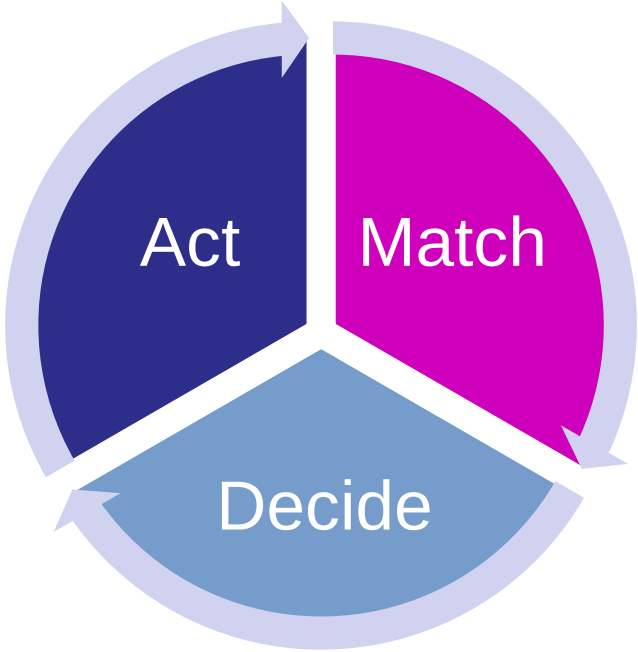
The Working Memory (WM)

- WM
  - Mother(*Rajmata_Jijabai*, *Chha. Shivaji*)
  - Spouse(Rajmata_*Jijabai, Shahaji)*
- Knowledge Base
  - PR1: Spouse(x, y) ∧ Mother(x, z) ⇒ Father(y, z)
- Interpreter
  - Match the variables
  - Decide rule to fire
  - Add: Father(*Shahaji*, *Chha. Shivaji*)

Interpreter / Inference Engine

- Passes through three step recognize-act cycle

Add new data to WM or delete old (and unnecessary) data, as suggested by the fired rule from the WM



Match the variables of the antecedents of a rule, kept in a knowledge base, with the data recorded in the WM.

Decide which rule to fire by designing a set of strategies for resolving the conflict regarding firing of the rules.

Conflict Resolution Process

- Identifying which rule to fire.

- Design a Rule-Set:
  - **Deterministic:** construct a rule-set where only *one rule is firable* at any instant of time
  - **Non-deterministic:** in real world contain, where *more than one rule* is firable at any instant of time.

- Performance: depends on
  - **Sensitive:** if the system can *respond quickly* to the change in environment, reflected by the new contents of the WM.
  - **Stability:** showing *continuity* in the line of reasoning.

Example

- Problem: sorting a string composed of the letters a, b, and c.
- Working memory: cbaca
- Production Rule set:
  - ba ⇒ ab
  - ca ⇒ ac
  - cb ⇒ bc

Trace of production System

WM: cbaca

Production Rule set:

ba ⇒ ab
ca ⇒ ac
cb ⇒ bc

Iteration	WM	Conflict Set	Rule Fired
1	cbaca	1, 2, 3	1
2	cabca	2	2
3	acbca	3, 2	2
4	acbac	1, 3	1
5	acabc	2	2
6	aacbc	3	3
7	aabcc	Ø	Halt

Conflict Resolution Strategies

- Refractoriness:
  - The same rule should not be fired more than once.
- Recency:
  - Uses most recent elements of the WM for instantiating one of the rules.
- Specificity:
  - The rule with more number of condition clauses be fired.
  - KB:
    - PR1: Bird(x) → Fly(x)
    - PR2: Bird(x), Not emu(x) → Fly(x)

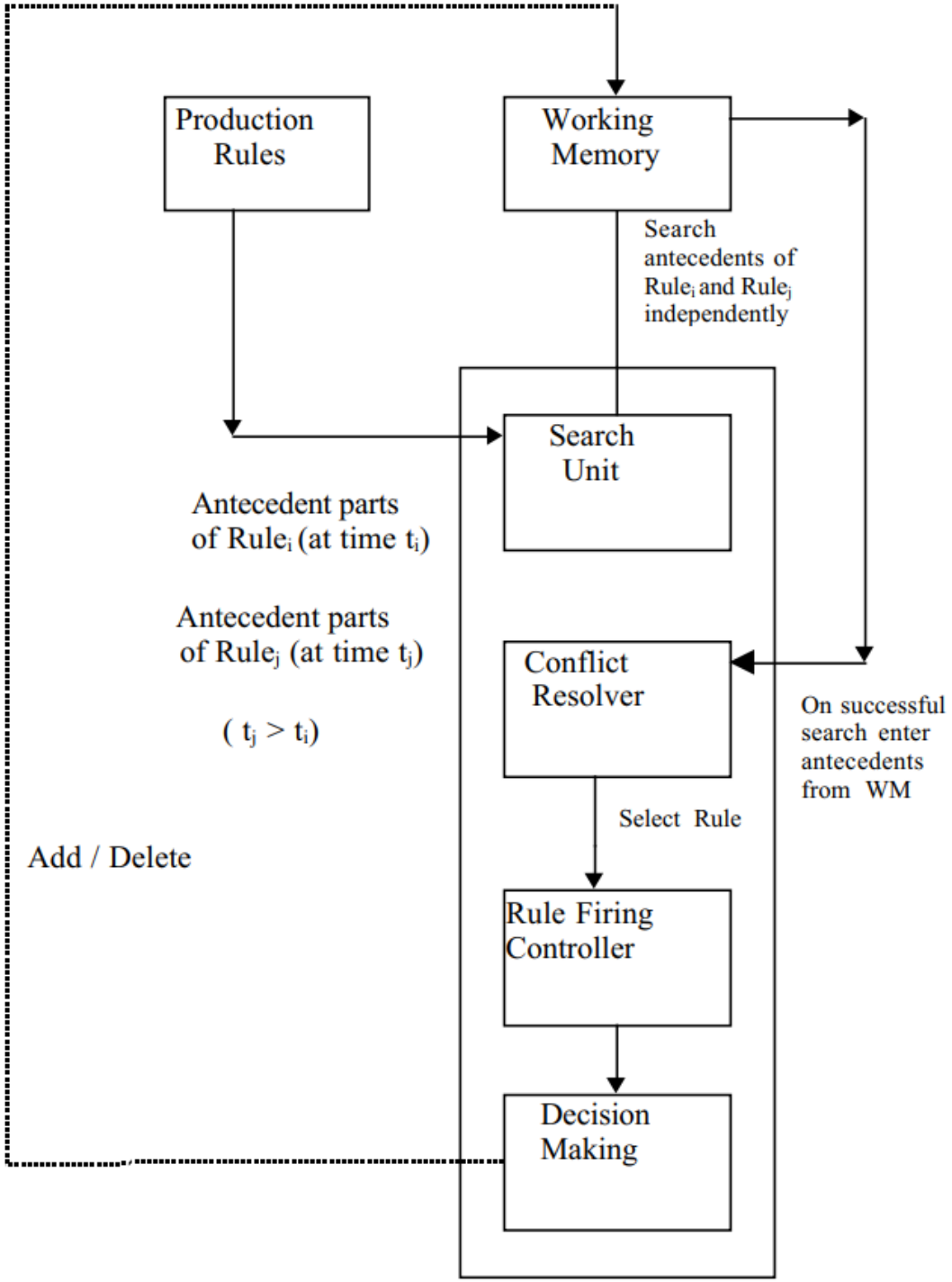
  WM:

  Bird(*parrot*),

  Not emu(*parrot*)

  - The second rule should be fired

Architecture

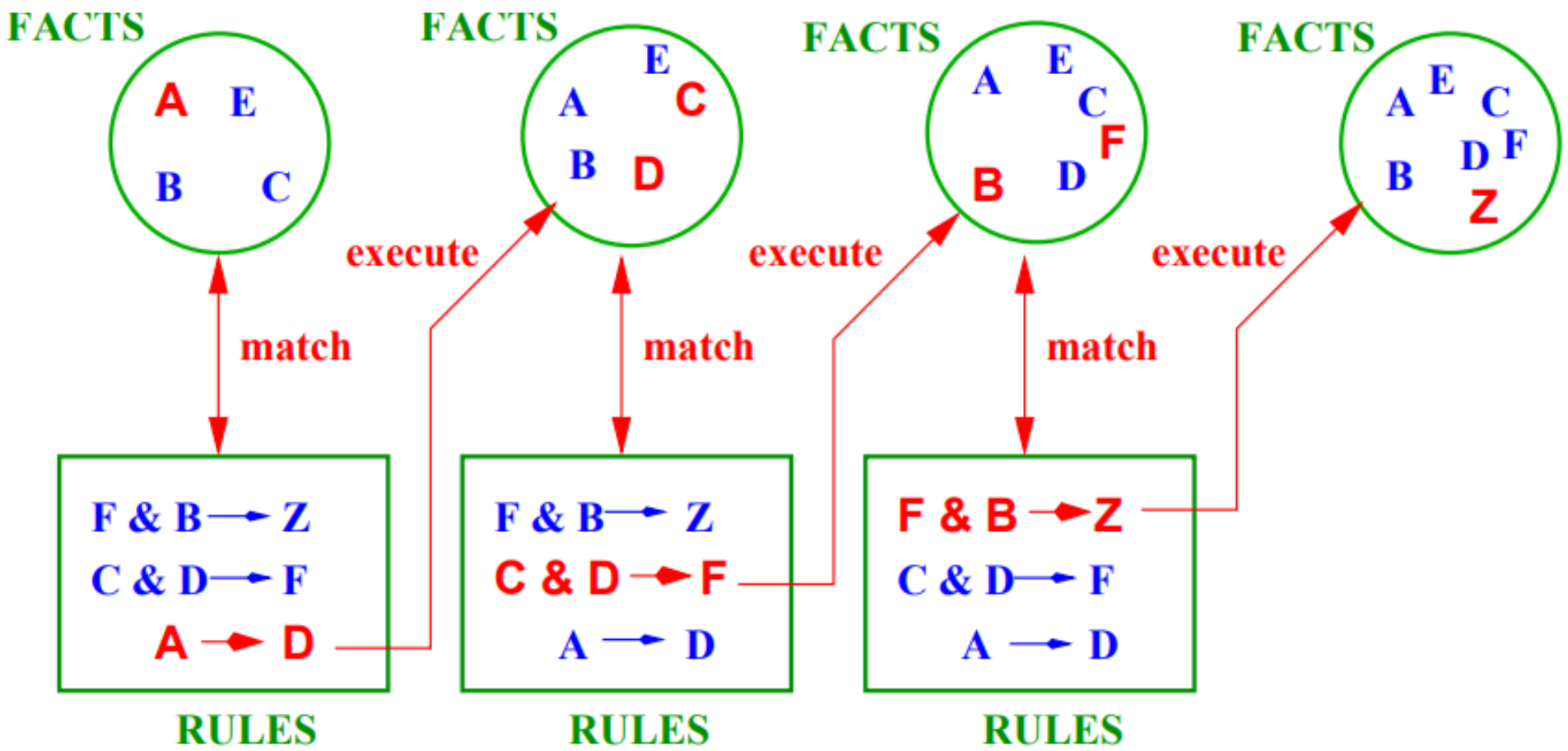


- Production Rules
- Working Memory
- Control Unit / Interpreter
  - Search Unit
  - Conflict Resolver
  - Rule Firing Controller
  - Decision Making

Inference Procedure

- Forward Chaining
  - Data-driven inference
    - Starting from the current state
    - Matching the condition of the rules (the IF parts)
    - Performing the corresponding actions (the THEN parts)
    - Update the WM
- Backward Chaining
  - Goal-driven inference:
    - looking at the WM to see if the sub-goal states already exist there.
    - If not, the actions (the THEN parts) of the rules that will establish the sub-goals are identified
    - new sub-goals are set up for achieving the premises of those rules (the IF parts).

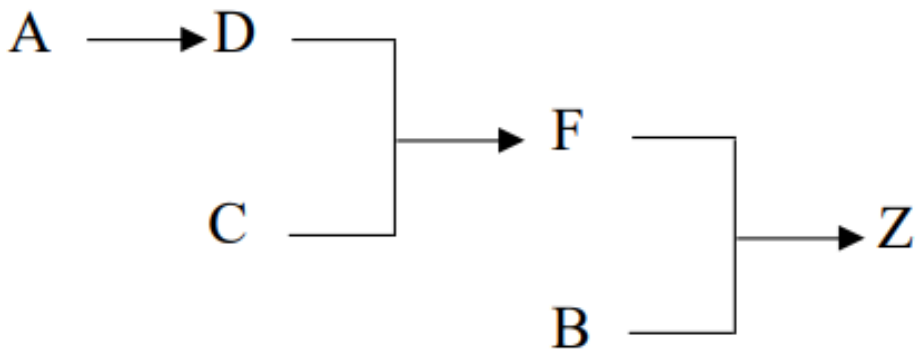
Forward Chaining



- The process continues until no more rules can be applied, or some cycle limit is met

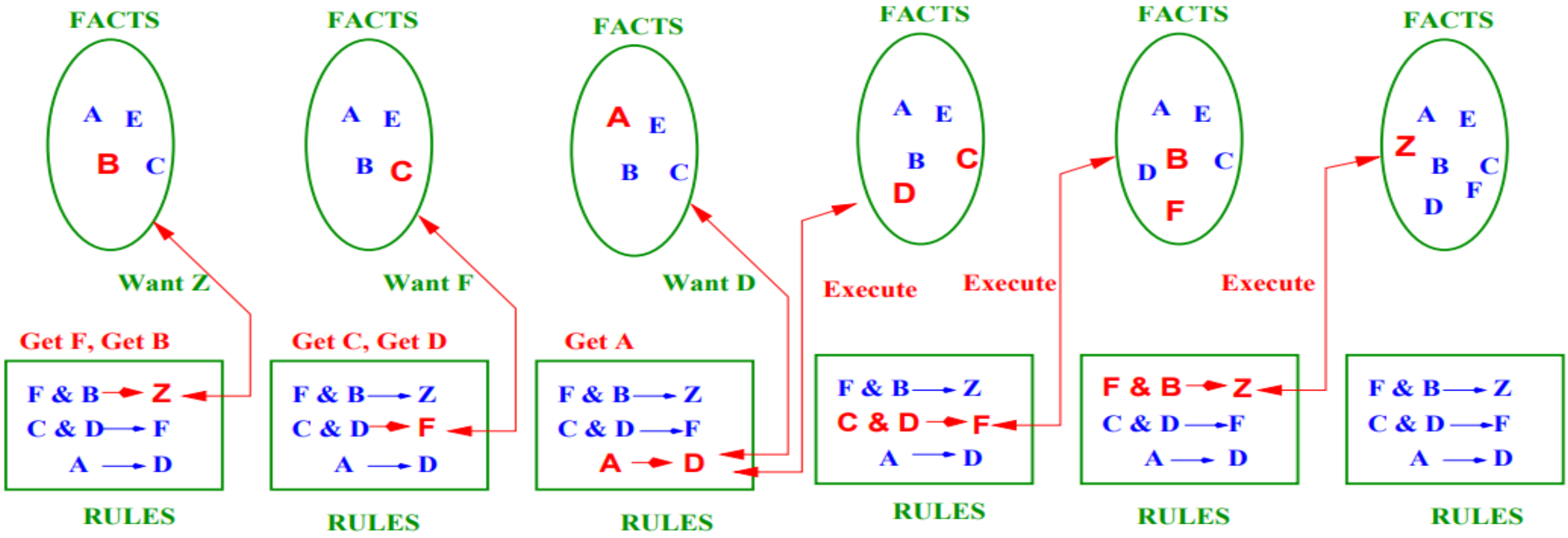
The Forward Inference Chain

- In this example there are no more rules, so we can draw the inference chain:



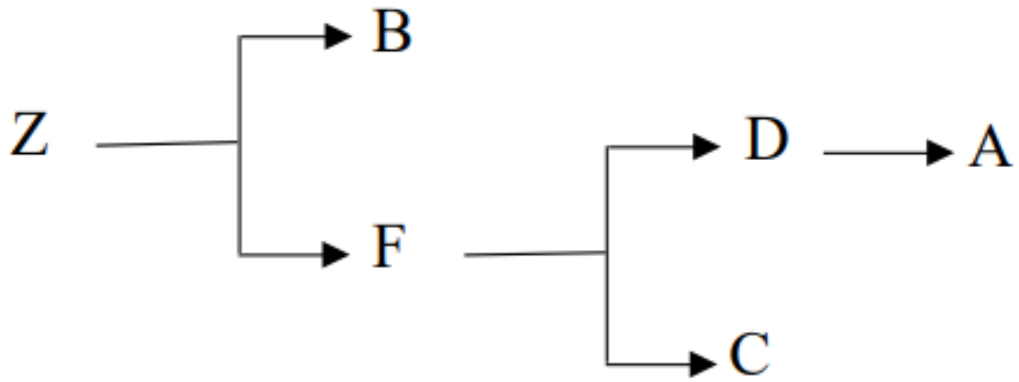
- This seems simple enough, but in this case we only had a few initial facts, and a few rules
- Generally, things will not be so straight forward.
- Disadvantages:
  - Many rules may be applicable at each stage – so how should we choose which one to apply next at each stage?
  - The whole process is not directed towards a goal, so how do we know when to stop applying the rules?

Backward Chaining



The Backward Inference Chain

- The first part of the chain works back from the goal until only the initial facts are required, at which point we know how to traverse the chain to achieve the goal state.



- Advantage:
  - The search is goal directed, so we only apply the rules that are necessary to achieve the goal.
- Disadvantage
  - A goal has to be known. Fortunately, most AI systems we are interested in can be formulated in a goal based fashion.

Example

Production Rules:

PR1: IF the ignition key is on
AND the engine won't start
THEN the starting system (including battery) is faulty

PR2: IF the starting system (including battery) is faulty
AND the headlights work
THEN the starter is faulty

PR3: IF the starting system (including battery) is faulty
AND the headlights not work
THEN the battery is dead

PR4: IF the voltage test on the ignition switch shows 1 to 6 volts
THEN the wiring between the ignition and the solenoid is OK

PR5: IF the wiring between the ignition and the solenoid is OK
THEN replace the ignition switch

WM:

The ignition key is on

The engine won't start

The headlights work

The voltage test on the solenoid shows 1 to 6 volts

Example

Production Rules:

PR1: IF the ignition key is on
AND the engine won't start
THEN the starting system (including battery) is faulty

PR2: IF the starting system (including battery) is faulty
AND the headlights work
THEN the starter is faulty

PR3: IF the starting system (including battery) is faulty
AND the headlights not work
THEN the battery is dead

PR4: IF the voltage test on the ignition switch shows 1 to 6 volts
THEN the wiring between the ignition and the solenoid is OK

PR5: IF the wiring between the ignition and the solenoid is OK
THEN replace the ignition switch

WM:

A: The ignition key is on

B: The engine won't start

C: The headlights work

D: The voltage test on the solenoid shows 1 to 6 volts

Example

Production Rules:

PR1: IF the ignition key is on
AND the engine won't start
THEN the starting system (including battery) is faulty

PR2: IF the starting system (including battery) is faulty
AND the headlights work
THEN the starter is faulty

PR3: IF the starting system (including battery) is faulty
AND the headlights not work
THEN the battery is dead

PR4: IF the voltage test on the ignition switch shows 1 to 6 volts
THEN the wiring between the ignition and the solenoid is OK

PR5: IF the wiring between the ignition and the solenoid is OK
THEN replace the ignition switch

PR1: $A \wedge B \Rightarrow E$

PR2: $E \wedge C \Rightarrow G$

PR3: $E \wedge \sim C \Rightarrow I$

PR4: $D \Rightarrow F$

PR4: $F \Rightarrow H$

Example – Forward Chaining

Production Rules:

PR1: $A \wedge B \Rightarrow E$

PR2: $E \wedge C \Rightarrow G$

PR3: $E \wedge \sim C \Rightarrow I$

PR4: $D \Rightarrow F$

PR5: $F \Rightarrow H$

WM:

A

B

C

D

Goal:

Infer all possible Facts

Conflict Resolution Strategy:

Refractoriness

Iteration	WM	Conflict Set (Queue)	Rule Fired	Action
1	ABCD	1, 4	1	PR1 add E
2	ABCDE	4, 2	4	PR1 is no longer applicable, PR4 add F
3	ABCDEF	2, 5	2	PR4 is no longer applicable, PR2 add G
4	ABCDEFG	5	5	PR2 is no longer applicable, PR5 add H
5	ABCDEFGH			No applicable rules (all are used!)

Example – Backward Chaining

Production Rules:

PR1: $A \wedge B \Rightarrow E$

PR2: $E \wedge C \Rightarrow G$

PR3: $E \wedge \sim C \Rightarrow I$

PR4: $D \Rightarrow F$

PR5: $F \Rightarrow H$

WM:

A

B

C

D

Goal:

$H \wedge I$

Conflict Resolution Strategy:

Refractoriness

Iteration	WM	Conflict Set	Rule Fire	Action
1	ABCD	5 (F Not in WM), 4 (D in WM)	4	PR4 add F
2	ABCDF	5 (F in WM)	5	PR5 add H
3	ABCDFH	3 (E and ~C Not in WM), 1 (A and B in WM)	1	PR1 add E
4	ABCDFHE	3 (~C Not in WM),		I is not supported, H ∧ I rejected

Example2 – Forward Chaining

Production Rules:

PR1: $P \wedge Q \Rightarrow GOAL$

PR2: $R \wedge S \Rightarrow P$

PR3: $W \wedge R \Rightarrow P$

PR4: $T \wedge U \Rightarrow P$

PR5: $V \Rightarrow S$

PR6: $START \Rightarrow V \wedge R \wedge Q$

WM:

START

Goal:

$P \wedge Q \Rightarrow GOAL$

Conflict Resolution Strategy:

Choose rule that has fired least recently

Example2 – Forward Chaining

Production Rules:

PR1: $P \wedge Q \Rightarrow GOAL$

PR2: $R \wedge S \Rightarrow P$

PR3: $W \wedge R \Rightarrow P$

PR4: $T \wedge U \Rightarrow P$

PR5: $V \Rightarrow S$

PR6: $START \Rightarrow V \wedge R \wedge Q$

WM:

START

Goal:

$P \wedge Q \Rightarrow GOAL$

Conflict Resolution Strategy:

Choose rule that has fired least recently

Iteration	WM	Conflict Set	Rule Fired	Action
1	START	6	6	PR6 add V, R, Q
2	START, V, R, Q	6, 5	5	PR5 add S
3	START, V, R, Q, S	6, 5, 2	2	PR2 add P
4	START, V, R, Q, S, P	6, 5, 2, 1	1	PR5 add H
5	START, V, R, Q, S, P, GOAL	6, 5, 2, 1	Halt	Stop

Example2 – Backward Chaining

Production Rules:

PR1: $P \wedge Q \Rightarrow GOAL$

PR2: $R \wedge S \Rightarrow P$

PR3: $W \wedge R \Rightarrow P$

PR4: $T \wedge U \Rightarrow P$

PR5: $V \Rightarrow S$

PR6: $START \Rightarrow V \wedge R \wedge Q$

WM:

START

Goal:

$P \wedge Q \Rightarrow GOAL$

Conflict Resolution Strategy:

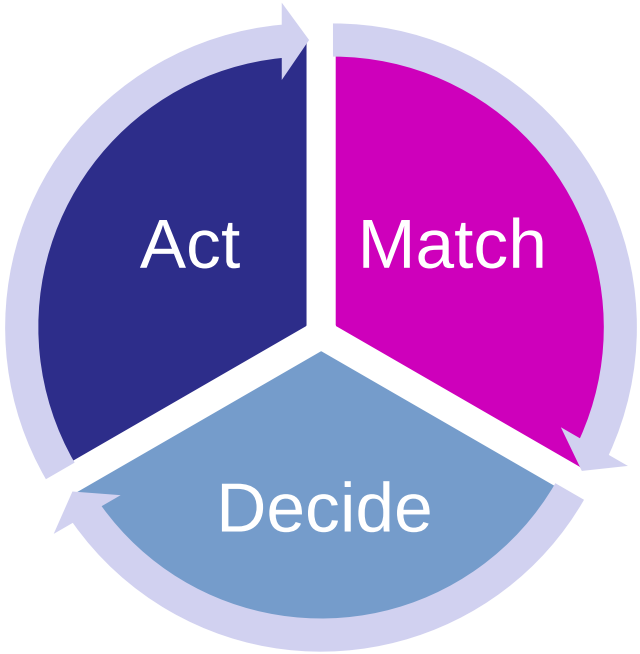
Choose rule that has fired least recently

Iteration	WM	Conflict Set	Rule Fired	Action
1	GOAL	1	1	
2	GOAL, P, Q	1, 2, 3, 4	2	
3	GOAL, P, Q, R, S	1, 2, 3, 4, 5	3	
4	GOAL, P, Q, R, S, W	1, 2, 3, 4, 5	4	
5	GOAL, P, Q, R, S, W, T, U	1, 2, 3, 4, 5	5	
6	GOAL, P, Q, R, S, W, T, U, V	1, 2, 3, 4, 5, 6	6	
7	GOAL, P, Q, R, S, W, T, U, V, START	1, 2, 3, 4, 5, 6	Halt	

Interpreter / Inference Engine

- Passes through three step recognize-act cycle

Add new data or delete old to WM



Match the rule antecedent variables with WM elements.

Decide which rule to fire from conflict set

Problems with the Recognize-Act Cycle

- How do we know that our production system is Consistent?
  - We need a *Reason Maintenance System*
- How do we know which Global Strategy for rule choice to apply?
  - We need to choose between *Forward and Backward* Chaining
- How do we know which Local Strategy for rule choice to apply?
  - We need a *Conflict Resolution System*
- How do we Maximize Efficiency as the Complexity increases?
  - We can use the *Rete Match Algorithm*

Reason Maintenance System

- PR1: IF *Raining* ∧ *Outside* ∧ ¬*Has_Umbrella* THEN *Wet*
- Facts in WM: *Dry*, *Outside*, *Raining*

- Rule to generate a new fact: *Dry*, *Outside*, *Raining*, *Wet*
- This is now an *inconsistent* set of facts.
- *Reason Maintenance System requires* into our system to deal with the inconsistencies.

Forward or Backward Reasoning?

- Are there more possible start states or goal states?
- Do we require the system to justify its reasoning?
  - Prefer the direction that corresponds more closely with the way that users think.
- What kind of events trigger problem solving?
  - If it is the *arrival of a new fact*, then forward chaining makes sense.
  - If it is a *query to which a response is required*, then backward chaining will be more natural.
- In which direction is the branching factor greatest?
  - Go in the direction with the lower branching factor.

Conflict Resolution

1. *Match* the rules against the known facts to see which rules can fire.
2. If more than one rule can fire, use a *Conflict Resolution* strategy to choose one.
3. *Fire* the chosen rule, updating the list of known facts.
4. Check the *Termination Criterion* and either stop or return to step 1.

- *Conflict Resolution:* can dramatically affect the solution

- Category
  - General conflict resolution
  - Problem specific conflict resolution.

General Conflict Resolution Strategies

- Five most common strategies are:
  1. Delete instantiations of rules that have already fired.
     - Prevents obvious endless loops of the same rules
  2. Order instantiations by the generation age of all the elements. Prefer the youngest.
     - New elements are more likely to describe the current situation.
  3. Compare the generation age of the elements in working memory which match the first condition of the rules. Prefer the youngest.
     - As for strategy 2, but may be more efficient.
  4. Prefer the most specific rules (i.e. those with the most pre-conditions).
     - This catches any exceptions/special cases before applying more general rules.
  5. Random choice.
     - Very easy to compute.

Specific Conflict Resolution Example

Consider the following set of rules:

PR1:	IF: engine does not turn AND battery is not flat
	THEN: ask user to test starter motor
PR2:	IF: there is no spark
	THEN: ask user to check the points
PR3:	IF: engine turns AND engine does not start
	THEN: ask user to check the spark
PR4:	IF: engine does not turn
	THEN: ask user to check the battery
PR5:	IF: battery is flat
	THEN: ask user to charge battery AND EXIT

WM: “engine does not turn” and “battery is not flat”

The conflict set is: { ⟨ R1, engine does not turn, battery is not flat ⟩, ⟨ R4, engine does not turn ⟩ }

We can see that our general conflict resolution strategy 4 would work well here.

Problem Specific Conflict Resolution: Extra Conditions

- Add extra conditions to the rules to avoid the conflicts.
- In our previous example we might modify R1 to give:

R1: IF: haven't already tested starter motor
AND engine does not turn
AND battery is not flat
THEN: ask user to test starter motor

- Disadvantages
  - Mixture of heuristics and factual knowledge.
  - Large knowledge bases will not be easily maintainable.

Problem Specific Conflict Resolution: Meta-Rules

- To avoid mixing the conflict resolution heuristics with the rule base by separating the object level knowledge from the meta-level knowledge.
- In our previous example we might supplement rule R1 with a meta-rule to give:

PR1:	IF: engine does not turn
	AND battery is not flat
	THEN: ask user to test starter motor
META-RULE M1:	IF: haven't already tested starter motor
	THEN: select R1

- Naturally this will increase the overall number of rules
- But it does separate the factual and heuristic knowledge and makes the knowledge base easier to maintain.

RBS Issue

- Disadvantage of RBS – large computational requirement to perform match of condition (LHS of rule) – determine if all instantiations of rules are satisfied by the content of the WM
- **O**(comparison to check the satisfaction of a rule) = $\mathbf{|WM|^{|CE|}}$

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

  |CE| - number of condition elements in a rule
- Computational effort
  - Match – 80-90%

RETE Match Algorithm

- Spelt as “Ree-tee Algorithm”
- Invented by Charles L. Forgy (1974)
  - Rete means the anatomical network of blood vessels and nerve fibers.
- Matching the condition of production rules with WM elements
- Determine which of the rules should fire based on data stored WK
- Intended to improve the speed of forward-chaining
- The network is in the form of an augmented dataflow network
- The Rete algorithm compiles the LHS of the production rules into a **discrimination network**
- **Changes to WM** are **input** to the network
- The network reports **changes to the Conflict Set (CS)** - **output** of the network

Match Steps

- What is the best way to determine the changes to the CS that result from changes to the WM?
- 2 ways of incorporating state information in the Match step to gain efficiency
- **Memory support**
  - Provides knowledge about which WMEs partially satisfy each individual Conditional Elements (CE)
  - An indexing scheme indicates which subset of WM partially matches each CE → ***alpha memory***
- **Condition relationship**
  - Provides knowledge about the interaction of CE within a rule and the partial satisfaction of rules → ***beta memory***

The Rete Algorithm

- Uses a rooted acyclic directed graph
  - Nodes: represent patterns (exception of the root)
  - Root to leaves paths : represent left-hand sides of rules
- Nodes:
  - Root node
  - One-input pattern nodes (for each condition): Alpha Node/Match Node
  - Two input join nodes (join two condition): Beta Node/Merge Node
- Keeps up to date the information associated with the nodes in the graph.
  - Fact is added or removed from WM
  - A token representing that fact and operation is entered at the root of the graph and propagated to its leaves modifying as appropriate the information associated with the nodes.

Matching

- The condition/premise patterns in the rules need to be matched with the known facts.
- Consider, a typical rule (about the value of horses) that matches a set of facts:

Rule

IF:	x is a horse
	x is the parent of y
	y is fast
THEN:	x is valuable

Facts

Comet	is-a	horse
Prancer	is-a	horse
Comet	is-parent-of	Dasher
Comet	is-parent-of	Prancer
Prancer	is	fast
Dasher	is-parent-of	Thunder
Thunder	is	fast
Thunder	is-a	horse
Dasher	is-a	horse

- In general, variables (e.g. x and y) in the rules which stand for arbitrary objects.
- We need to find bindings for them so that the rule is applicable.

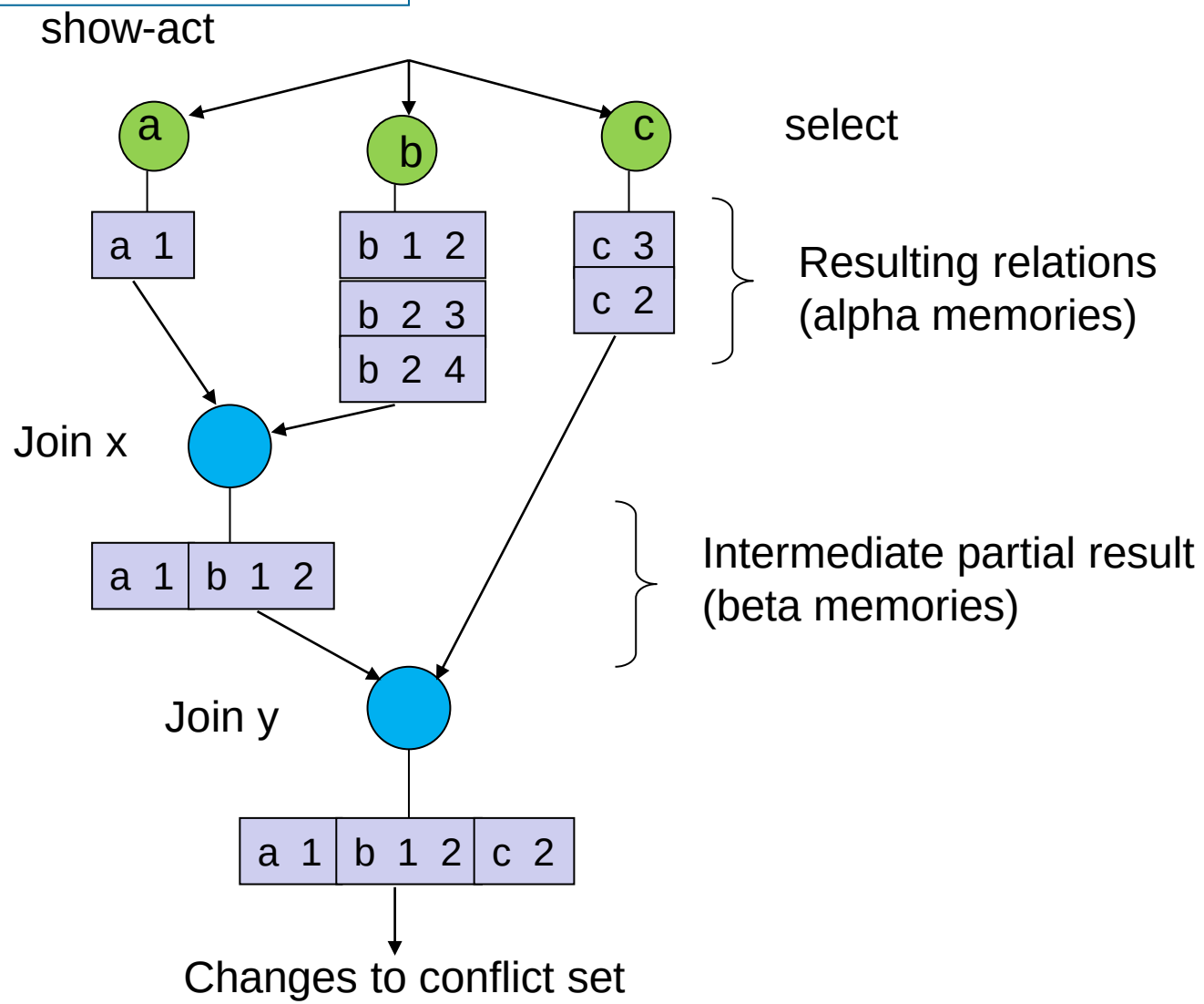
Binding

Bindings for "*x* is-a horse"	Bindings for "*y* is fast"	Bindings for "*x* is a parent of *y*"
x = Comet	*y* = Prancer	*x* = Comet , *y* = Dasher
x = Prancer	*y* = Thunder	*x* = Comet , *y* = Prancer
x = Thunder		*x* = Dasher , *y* = Thunder
x = Dasher		

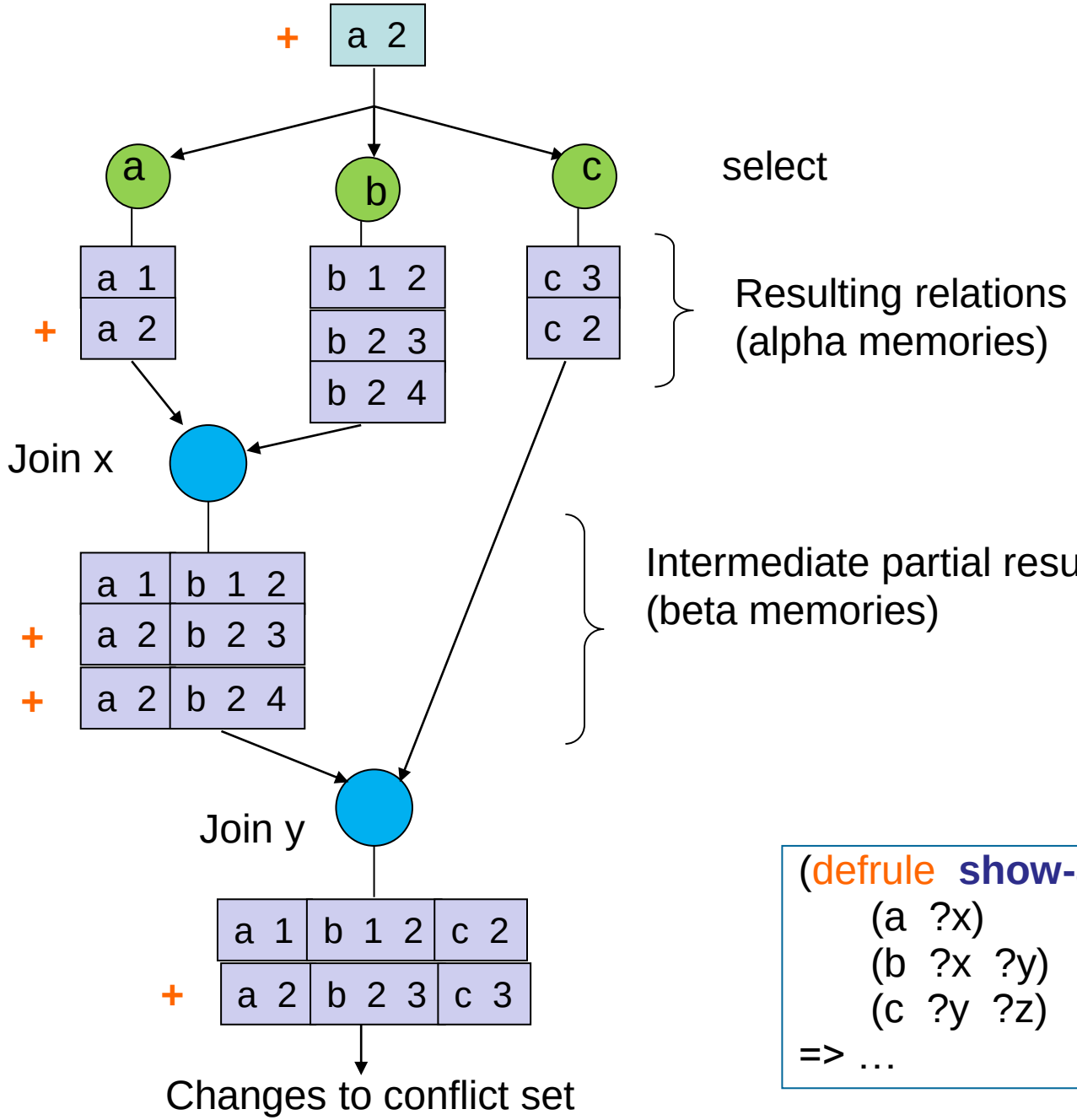
- From these we can deduce that there are two possible bindings applicable to the rule:
  - x = Comet and y = Prancer
  - x = Dasher and y = Thunder
- The rule then tells us "x is valuable"
  - Comet is valuable
  - Dasher is valuable

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

Initial state of RETE Network

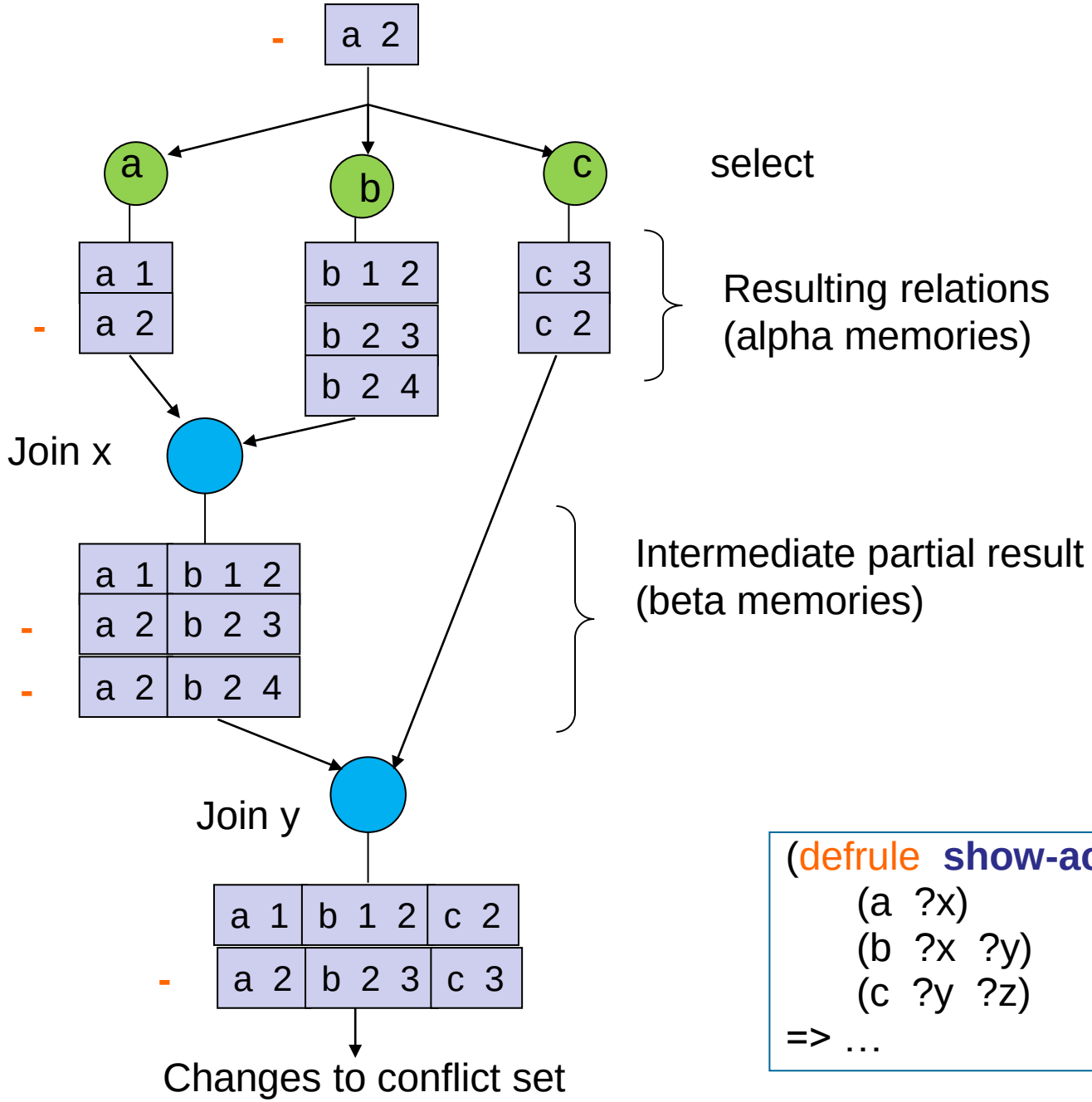


Activity of the RETE Match during an Addition




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

Activity of the RETE Match during a Deletion



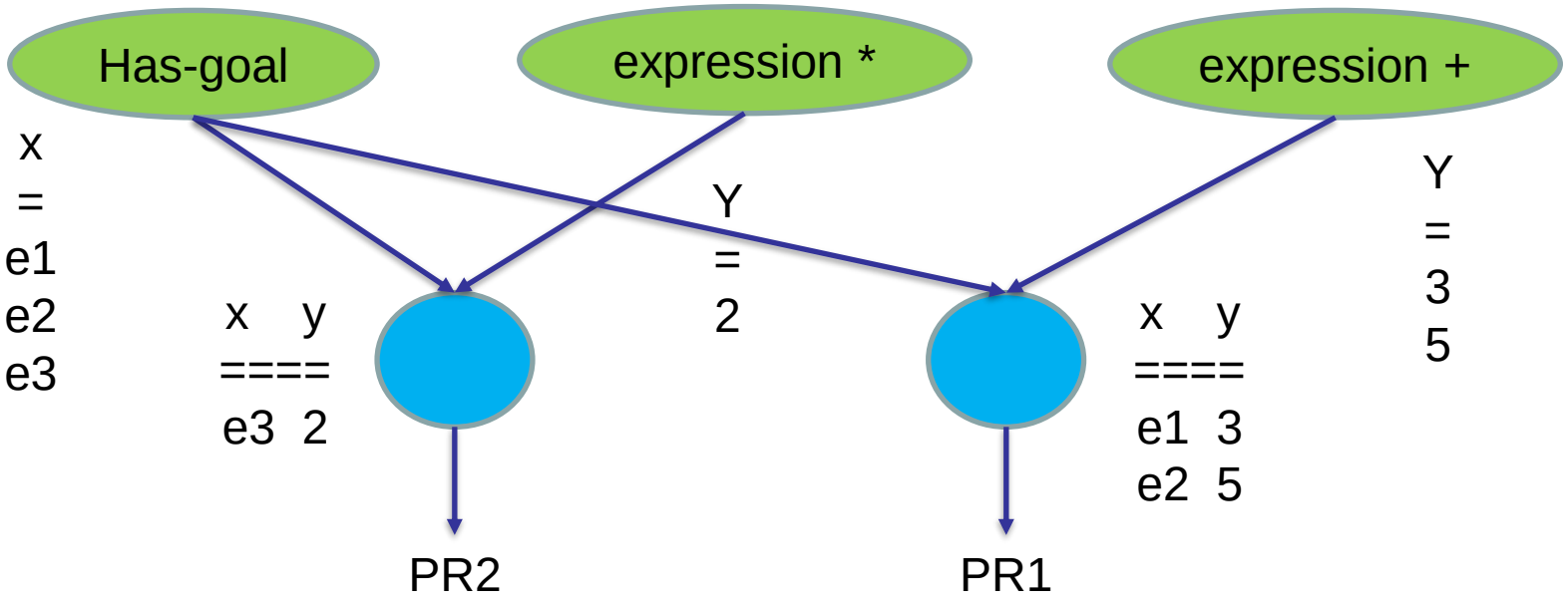

```
(defrule  show-act
      (a  ?x)
      (b  ?x  ?y)
      (c  ?y  ?z)
=> …
```

Example

- PR1: (has-goal ?x simplify), (expression ?x 0 + ?y) ⇒ ....
- PR2: (has-goal ?x simplify), (expression ?x 0 * ?y) ⇒ ....

- WM:

(has-goal e1 simplicity)
(expression e1 0 + 3)
(has-goal e2 simplicity)
(expression e2 0 + 5)
(has-goal e3 simplicity)
(expression e3 0 * 2)



Example

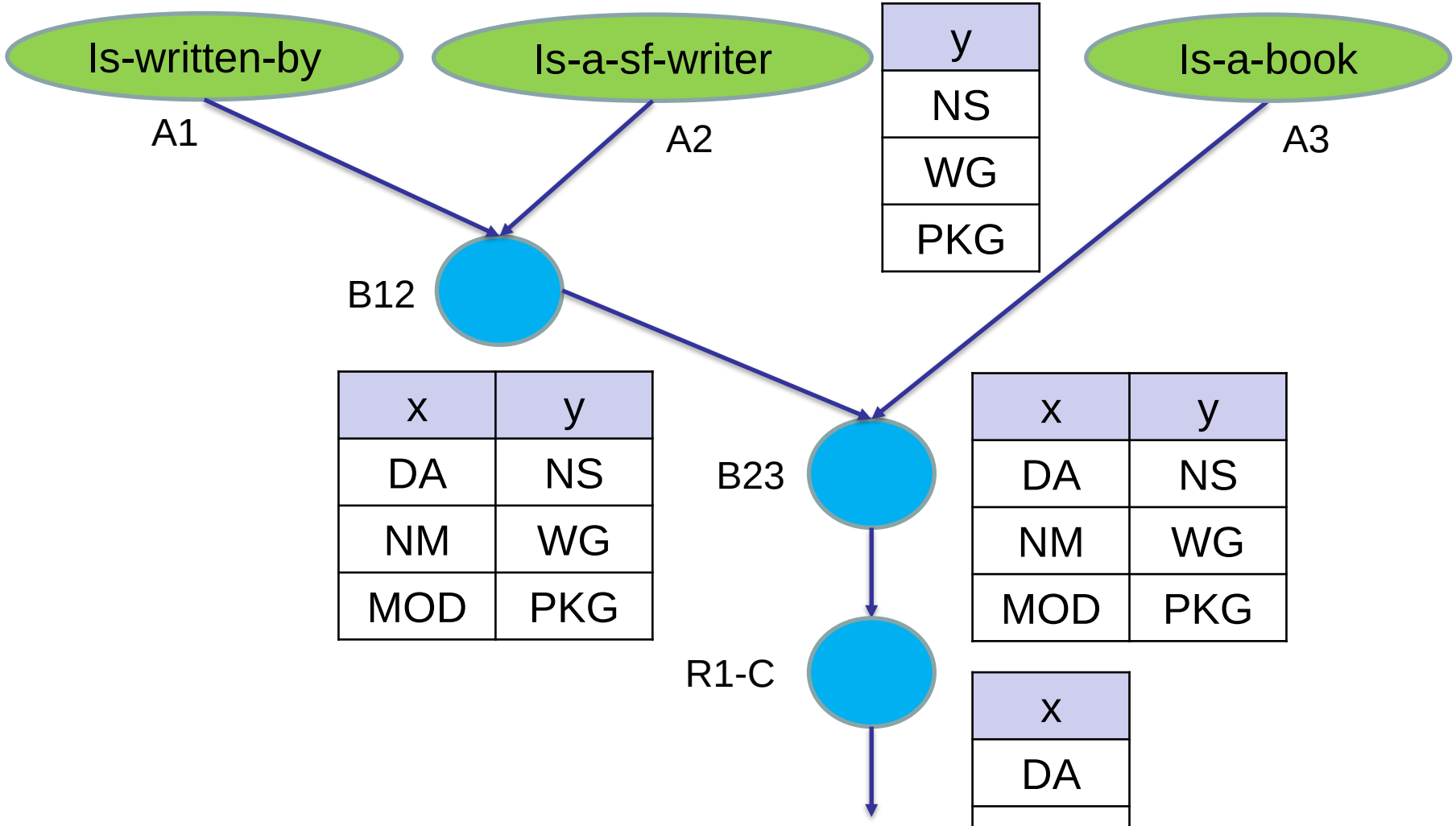
- PR1: IF (?x is-written-by ?y), A1
  (?y is-a science-fiction-writer), A2
  (?x is-a book), A3
  THEN (?x is-a science-fiction-book) R1-C

WM:

TheDiamondAge is-written-by NealStephenson
Neuromancer is-written-by WilliamGibson
NealStephenson is-a science-fiction-writer
Neuromancer is-a book
TheDiamondAge is-a book
MazeOfDeath is-written-by PhilipKDick
WilliamGibson is-a science-fiction-writer
PhilipKDick is-a science-fiction-writer
PlayerOfGames is-a book
Ubik is-a book
MazeOfDeath is-a book
PlayerOfGames is-written-by IainBanks

Rete Network

x	y
DA	NS
NM	WG
MOD	PKG
POG	IB



y
NS
WG
PKG

x
NM
DA
POG
UB
MOD

x	y
DA	NS
NM	WG
MOD	PKG

x	y
DA	NS
NM	WG
MOD	PKG

x
DA
NM
MOD

Example

- PR1: IF (?x is-written-by ?y), A1
  (?y is-a science-fiction-writer), A2
  (?x is-a book), A3
  THEN (?x is-a science-fiction-book) R1-C

PR2: IF (?x is-written-by ?y), A1
(?y is-a science-fiction-writer), A2
(?x is-a short-story), A4
THEN (?x is-a science-fiction- short-story) R2-C

WM:

TheDiamondAge is-written-by NealStephenson
Neuromancer is-written-by WilliamGibson
NealStephenson is-a science-fiction-writer
Neuromancer is-a book
TheDiamondAge is-a book
MazeOfDeath is-written-by PhilipKDick
BurningChrome is-a short-story
JohnnyMnemonic is-a short-story
SecondVariety is-written-by PhilipKDick
SecondVariety is-a short-story

WilliamGibson is-a science-fiction-writer
PhilipKDick is-a science-fiction-writer
PlayerOfGames is-a book
Ubik is-a book
MazeOfDeath is-a book
PlayerOfGames is-written-by IainBanks
BurningChrome is-written-by WilliamGibson
BluebeardEgg is-a short-story
JohnnyMnemonic is-written-by WilliamGibson
BluebeardEgg is-written-by MargaretAtwood

Example

- Given 2 water jugs, 4 liters and 3 liters. Neither has any measuring marks on it. There is a pump that can be used to fill the jugs. How can you get exactly 2 liters of water into 4-liter jugs?
- Consider:
  - u denote the content of 4L jugs
  - v denote the content of 3L jugs
- WM = {(0, 0)}
- Find list of PRs
- Solve using:
  - Forward Changing
  - Backward Changing

Example

A farmer wants to transfer his three belongings, a wolf, a goat and a cabbage, by a boat from the left bank of a river to its right bank. The boat can carry at most two items including the farmer. If unattended, the wolf may eat up the goat and the goat may eat up the cabbage. How should the farmer plan to transfer the items?

- The illegal states: (W,G || F,C) , (G,C || F,W), (F, W || G, C), (F, C || W, G)
  - F denote farmer
  - G denote goat
  - W denote wolf
  - || denote river

- WN = WGFC||
- Find list of PRs
- Solve using:
  - Forward Changing
  - Backward Changing

Summery

- Production systems are the simplest method for knowledge representation in AI.
- Subject-domain Experts, encoding knowledge in simple if-then rules.
- The efficiency depends mainly on the order of firing of rules and on the conflict resolution strategies.
- Selection of conflict resolution strategies is a significant issue in designing a production system.
- Rete Matching Algorithm increase the efficiency of PBS.