1

Knowledge Based Systems

(CM0377)

Lecture 11a

(Last modified 26th March 2001)

2

The RETE algorithm

- As you can imagine, a naïve implementation of a forward chaining system could become extremely inefficient, if there were many rules.
- The RETE algorithm is based around a clever structure that greatly reduces the processing required

3

Hypertext Webster Gateway: "Rete"

- From Webster's Revised Unabridged Dictionary (1913)
  - Rete \Re"te\, n. [L., a net.] (Anat.) A net or network; a plexus; particularly, a network of blood vessels or nerves, or a part resembling a network.

4

The basic idea

- Build a network whose input is working memory elements and whose leaf nodes are the rules of the system;
- When a new WM element is added, propagate it through the network to determine if it affects variable bindings for which each rule can fire. Similarly if a WM element is deleted.
  (WM element updates are most simply regarded as deletion followed by addition).
- Thus, maintain a list of rules that can fire & instantiations (variable bindings) for which each can fire, and *update* it each time a change is discovered in the Rete.

5

What goes in the network?

- The network comprises:
  - A *root node;*
  - A SELECT node for each pattern that appears on the left-hand sides of the rules;
  - For each rule, and for each condition of each rule, an α node connected to the appropriate SELECT node;
  - For each rule, a β (beta) node joining the α nodes for the first two conditions, then for each subsequent condition, a β node joining the previous β node and the α node for this condition; and
  - For each rule, a node connected to the final β node for that rule.
- With each kind of node is associated a particular relational database operation. A table is associated with each node.

6

SELECT nodes

- These nodes SELECT those WM elements which match their associated patterns, without checking consistency with other conditions in the rules.

7

α nodes and β nodes

- α nodes:
  - These PROJECT the selected WM elements, renaming as appropriate, onto a relation whose columns are the variables of the associated condition. No consideration has yet been taken of interrelationship between conditions.
- β nodes:
  - These combine conditions using a JOIN operation in a step-wise manner.
  - Thus, find variable bindings consistent with first 2 conditions in a rule, then combine this with 3rd condition, etc.
- Even if we 'fudged' by having a single α node for each pattern, with appropriate renaming going on, why would separate β nodes be needed for each rule?

8

Rule nodes

- These have an associated PROJECT operation which maps the final β node's variable bindings onto a table whose columns correspond to variables occurring in the rule's *action*.

9

Propagation of WM elements

- When a new WM element arrives, it is simply propagated through the network, as illustrated in the accompanying figures.
- *Initialise* network by adding WM elements of initial state one at a time.

10

Things to note

- This technique clearly improves speed of conflict set generation (we won't try to analyse it …).
- This technique uses plenty of memory, so is not necessarily always feasible.
- Rete provides a *conflict set.* Conflict resolution can then be performed separately.
- With a Rete network, evaluation of rule conditions never need occur: the network computes which rules *could* fire.

11

Things to note (Ctd.)

- Very (time-)efficient implementations of a Rete network can be achieved by generating code *for each* network node (not just for each node *type)* that performs the appropriate operation (SELECT, JOIN or whatever).
- We have ignored deletion of WM elements. This is a fairly straightforward extension, conceptually, but note that quite a large amount of the network may need recomputation if a WM element that was introduced early on is deleted.

12

Example

The accompanying example implements the rules:

```
R1    IF ?x child-of ?y
      THEN ADD ?y parent-of ?x

R2    IF ?x parent-of ?y
      AND ?x listens-to ?z
      AND ?y is-musical true
      THEN ADD ?y listens-to ?z

R3    IF ?x parent_of ?y
      AND ?x lives-in flat
      THEN ADD ?x has-reputation nuisance
```

13

Example (ctd.)

We assume the following initial working memory state:

```
WM1   karen child-of martha
WM2   ellen child-of annabel
WM3   annabel listens-to bach
WM4   annabel lives-in house
WM5   martha lives-in flat
WM6   ellen is-musical true
WM7   billy child-of martha
```

14

Example (ctd.)

- Sequence of operations we choose to follow:
  - Create network
  - Add WM1
  - Add WM2
  - Add WM3
  - Add WM4
  - Add WM5
  - Add WM6
  - Add WM7
  - Fire R1 on ?x/karen, ?y/martha, adding *WM8: martha parent-of karen*
  - Fire R1 on ?x/ellen, ?y/annabel, adding *WM9: annabel parent-of ellen*
  - Fire R1 on ?x/billy, ?y/martha, adding *WM10: martha parent-of billy*
  - Fire R3 on ?x/martha, adding *WM11: martha has-reputation nuisance*

15

Further reading ...

- For a detailed worked example with just one rule: P.H. Winston, Artificial Intelligence (3rd edition), pp. 148-161.
- If you're really interested, the original paper is: C.L. Forgy, Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem, in Artificial Intelligence Vol. 19 no. 1, pp. 17-37.