

Introduction to the Rete Algorithm

0 1 4,206

What is Rete?

The **Rete algorithm** is a pattern matching algorithm designed by Dr Charles L. Forgy of Carnegie Mellon University. Rete is a Latin word which means *net* [1]. It is a very efficient algorithm for matching facts against the patterns in rules. Understanding of the Rete algorithm will make one easier to understand why writing rules one way is more efficient than writing them another way.

Rules, Ruleset and Facts

A ruleset is nothing but a knowledge base consisting of one or more business rules (or simply rules). Every rule in the ruleset represents some knowledge. Rules are usually in the form of if-then or condition-action. As if-then rules are suitable for rete algorithm they are called rete rules. Here is a simple rete rule

If age > 60 then assign status = "Senior Citizen"

Where *if*, *then* and *assign* are keywords. *If part* represents condition and *then part* represents action. There may be more than one condition and in that case the conditions should be joined by logical operators. *Then part* represents one or more actions. Clearly the above example rule means that if a person's age is more than 60 then he is a senior citizen. If we want to check whether a person is senior citizen or not we need a data that is the person's age and this data is called fact. As the number of unique variables or definitions increases, so does the number of facts. For example,

If age > 60 and annual-income < 12000 then assign monthly-bonus = 2000

To execute the above rule we need two data or facts: one for age and one for annual-income. For checking the bonus of two persons we need two pair of data, and so on. This is a simple case and in real life situations we may have thousands of facts where the rule should operate on. And we may have thousands of different rules as well.

Need of Rete Algorithm

A complete rule-set should be given to the rule engine for further processing. The rule engine matches each rule (i.e. condition) in the ruleset with given facts to decide whether to fire (or execute) the rule or not. This is called pattern matching process and this process takes place repeatedly. In each cycle the list of facts may be modified: new facts may be added to the list or old facts may be removed from the list. These changes may cause previously unsatisfied patterns to be satisfied. Moreover during each cycle the set of rules satisfied must be

maintained and updated. In most of the cases, actions of the rules change only a few facts in the list. This is called temporal redundancy. If a rule engine checks each rule to direct the search for all the facts even if most of them are not modified then it will slow down the process. This unnecessary computation can be avoided by remembering what has already matched from cycle to cycle and then computing only the changes necessary for the newly added or removed facts. Rete algorithm does this work perfectly.

Rete Algorithm: Implementation

The Rete algorithm is implemented by building a network of nodes. It is designed in such a way that it saves the state of the matching process from cycle to cycle and re-computes changes only for the modified facts. The state of the matching process is updated only as facts are added and removed. If the facts added or removed are less in number then the matching process will be faster.

The Inference Cycle

Each rule has an inference cycle consisting of three phases: match, select and execute. In matching phase, the conditions of the rules are matched against the facts to determine which rules are to be executed. The rules whose conditions are met are stored in a list called agenda for firing. From the list of rules, one of the rules is selected to execute or fire. The selections strategy may depend on priority, recency of usage, specificity of the rule, or on other criteria. The rule selected from the list is executed by carrying out the actions in the right-hand-side of the rule. The action may be an assertion, executing a user defined or built-in function or executing a decision table, or otherwise. Note that the decision table engine is used to execute decision tables.

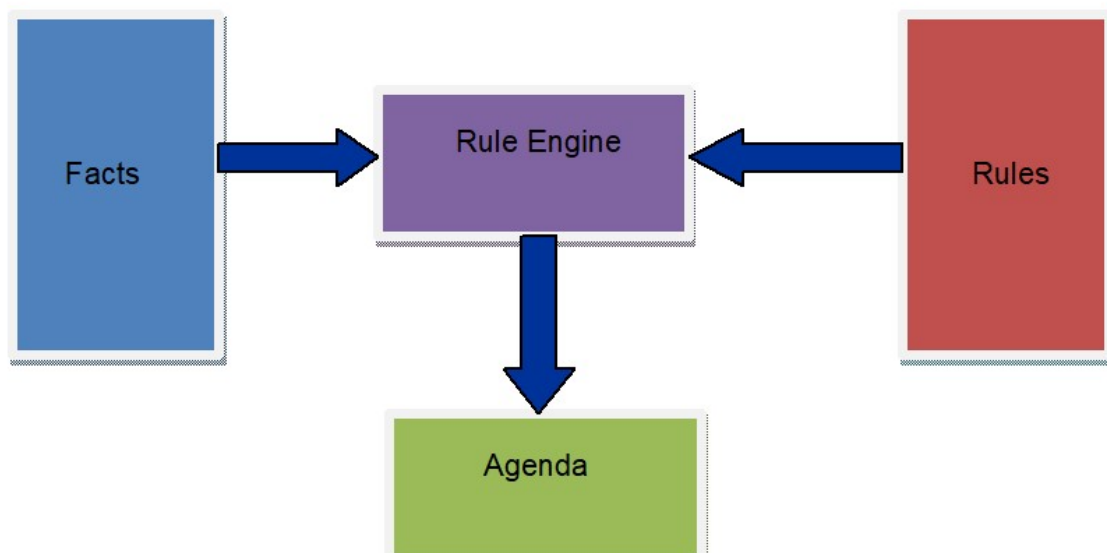


Figure1. Pattern matching: Rules and Facts

Building of the Rete Network

The Rete network is a direct acyclic graph consists of nodes representing patterns in the conditions of the rules. The nodes behave like filters, testing the incoming tokens, and sending only those that pass the test. The rete network consists of two parts: alpha network and beta network. Alpha network consists of nodes known as alpha nodes. Each alpha node has one input that defines intra-elements. Beta network consists of beta nodes where each node takes two inputs to define inter-element conditions.

The Rete network starts with the root node called *Rete Node*. Kind nodes follow the root node. There should be a kind node for each fact type. Alpha nodes are then created for each pattern and connected to the corresponding kind node. A condition, for example, $a > b$ or $b > c$ has two patterns, so two alpha nodes will be created: one for $a > b$ and one for $b > c$. Each alpha node is associated with a memory known as alpha memory. It is used to remember the facts matched. Alpha nodes are then joined by beta nodes. Beta node accepts only two inputs. So, if there are three alpha nodes then first two nodes will be joined by one beta node. Thereafter the output of this beta node and the third alpha node will be joined by another beta node. This way beta nodes support partial matching. Each beta node has a memory to store joined patterns.

Assertion of each fact creates a token. Initially the tokens enter the root node. The network then splits a branch for each token type. Each kind node gets a copy of the token and performs SELECT operation to select tokens of its kind. The kind node delivers a copy of the token node it receives to the alpha nodes. On receiving the token, the alpha nodes do a PROJECT operation and extract components from the token that match with the variables of the pattern. That is alpha node, basically, evaluates the conditions. Beta node then determines the possible cross product for a rule. Finally, actions in the rule will be executed.

Example

Suppose we have a rule,

If $\text{age} > 60$ or $\text{age} < 5$ or $\text{income} < 36000$ then $\text{concession} = 50/100$.

Assume that age is java variable and income is xml variable. The following rete network can be created to represent this rule.

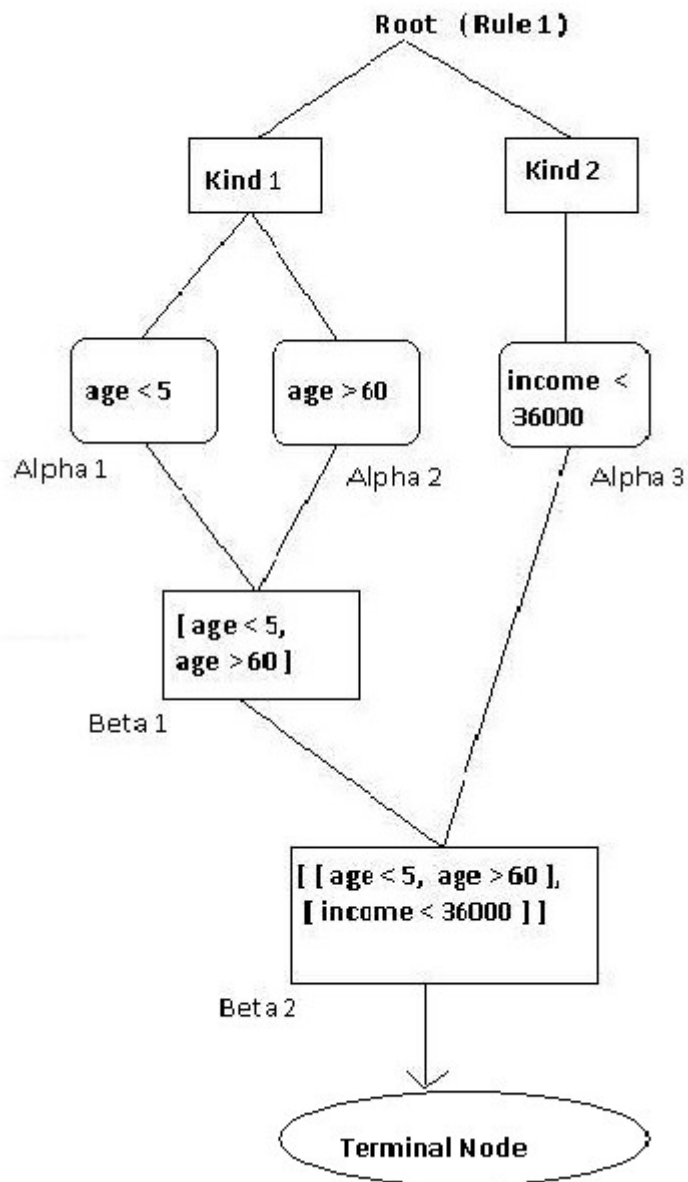


Figure 2. A simple Rete network for a single rule

In the above rete network, there are two kind nodes as there are two types of facts: java and xml. Kind 1 node represents java type and kind 2 node represents xml type. As there are three patterns: age>5, age>60 and income<36000 three alpha nodes will be created. Alpha 1 and alpha 2 representing the first two patterns are connected to kind1 and an alpha 3 representing the third pattern is connected to kind2. Now first two alpha nodes are joined by beta 1. The third alpha node and beta 1 joined by beta 2.

When a value for age enters the root a token will be created. Copies of this token will be passed to kind nodes. Kind1 will accept it as the fact type is java. This token will be passed onto alpha1 and alpha 2. If the value satisfies the constraint then the token will be passed onto beta 1 and then to beta 2. In the mean time value of income enters the root and then accepted by kind 2. Alpha 3 receive it and checks if the value satisfies the constraint, income < 36000.

If yes then it allows the token passing onto beta 2. If the fact, that is the values, match with the condition in the beta 2 then the rule will be added to a list for firing.

Advantages and Disadvantages

Advantages of Rete algorithm

The main advantage of Rete algorithm is speed as it takes the advantage of structural similarity in rules. Many rules often contain similar patterns or group of patterns. Rete algorithm pools the common components so that they need not be computed again.

Disadvantage of Rete algorithm

The main drawback of Rete pattern matching algorithm is that it is memory intensive. Saving the state of the system using pattern matches and partial matches considerable amount of memory. The space complexity of Rete is of the order of $O(RFP)$, where R is the number of rules, F is the number of asserted facts, and P is the average number of patterns per rule.

Poorly written rules run slowly. Moreover if the all the facts were to be compared against all the patterns then also the speed cannot be achieved. But this is only the worst case.

References

1. Patrick Henry Winston, Artificial Intelligence, Third Edition, Pearson Education.
2. Dan W. Patterson, Introduction to Artificial Intelligence And Expert Systems, PHI.
3. Stuart Russel & Peter Norvig, Artificial Intelligence- A Modern Approach, PHI.
4. Neil Madden, Optimizing Rete for Low-Memory, Multi-Agent Systems, In Proceedings of Game, 2003