



COMMON KNOWLEDGE WHITE PAPER

Rete Rules Extender

Making your business rules **your** business™





Business Rules

Pronunciation

Rete is usually pronounced either 'REET' or 'REE-tee'

The Rete Rules Extender provides an highly intuitive, understandable and maintainable format for representing business rules that evaluates rules using a sophisticated technique called the "Rete Algorithm".

The Rete Algorithm is a pattern-matching algorithm specifically designed for the efficient evaluation of a large number of complex, inter-related business rules.

The number of business rules used within an application or organisation can vary from a handful to many hundreds or perhaps thousands. For many organisations, the number of business rules may also increase over time due to factors such as growing customer numbers, increasing organisational complexity and additional product offerings. By utilising the Rete Rules Extender, scalable business rules solutions can be built that deliver performance largely independent of the number of rules represented within the system.

Background to Rete Rules Algorithm

History

Dr Forgy developed the Rete algorithm whilst at Carnegie Mellon University. The algorithm has become the basis of many popular Production Systems such as OPS5, CLIPS and Drools.

The Rete algorithm was designed by Dr. Charles L. Forgy around 1982 to facilitate efficient pattern-matching whilst evaluating a large number of rules. Research into expert systems had shown that in order for a system to exhibit intelligent behaviour it would need to acquire a large volume of knowledge and as such, the evaluation process would need to be highly scalable. In this context, knowledge is made up of two basic elements:-

- **facts** that have been asserted into the system, and
- **rules** that operate on these facts.

Whenever new facts are asserted into a system, many rules may need to be re-evaluated to determine which are satisfied by these facts. A brute-force implementation could simply re-evaluate all existing rules against the newly introduced facts. This approach would achieve a result, but would exhibit very poor performance as the number of rules and facts being considered increased. Dr. Forgy developed an algorithm that constructs a sophisticated network of nodes to represent the conditions associated with each rule and the dependencies between those conditions. This network maintains the state of partially-matched rules (conditions), and as new facts are introduced to the network, only those conditions dependent on the new facts need to be re-evaluated.

Representation of Rete Rules

Production Systems

A production system consists of a set of rules (in IF/THEN format), a database of current state information, and a rules interpreter. The rules interpreter uses a forward chaining loop to operate on the rules.

For rules to be evaluated using the Rete algorithm, they are typically represented in a *production system* format. Simply speaking, this format consists of a series of **USING-IF-THEN** structured statements as follows:

USING

```
Driver : DriverType  
Policy : PolicyType
```

IF

```
Driver.Age < 21
```

THEN

```
Policy.Risk = High
```





In traditional Production System terminology, The **USING** portion describes the classes of facts on which the rule is dependant, and provides instance names for those facts. These instance names are subsequently used to refer to the facts in the condition and action portions of the rule. In the above example, the rule is dependant on two types of facts, one being a **DriverType** which we refer to in the rule as *Driver*, and the other being a **PolicyType** which we refer to in the rule as *Policy*.

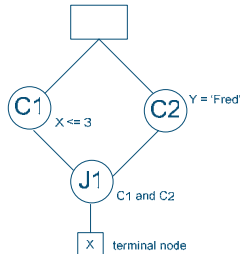
The **IF** portion of the statement (`Driver.Age < 21`) is called the *Condition*. The **THEN** portion (`Policy.Risk = High`) is called the *Action*. A single **USING-IF-THEN** statement is called a *Production*, and a group of productions is called a *Production-set*. In the Rete Rules Extender we refer to these as a *Rule* and a *Rule-set* respectively.

Considering the simplest example, it may be difficult to understand why these statements need to be assembled into a more complex structure like a network. However, when multiple rules become inter-related via terms that are shared between conditions within a rule or across rules, the rule logic can rapidly become complex and it is this complexity that is managed efficiently by the Rete Algorithm.

Once rules are represented in this **USING-IF-THEN** format, the Rete Algorithm can convert this structure into the required network of nodes, with nodes representing the conditions or combinations of the conditions associated with each rule. Where conditions contain similar expressions, the network can be further optimised by creating shared nodes within the network.

Rete Rules Mechanisms

Rete Node Network



A production system consists of a set of rules (in IF/THEN format), a database of current state information, and a rules interpreter. The rules interpreter uses a forward chaining loop to operate on the rules.

In the structure of a Rete network, a **condition node** represents a satisfied condition of a rule. A **join node** represents the combination of multiple conditions. When all conditions of a particular rule are satisfied then a **terminal node** is reached, indicating that the rule holds true when it is bound to the current contents of working memory.

At its most fundamental level, the Rete Algorithm works with a given set of rules which are stored in the Rete's *production memory*, and a given set of currently-known facts (objects) that are stored in the Rete's *working memory*. Working memory will include facts from the external world and also the current problem solving state of the rete network. The diagram below illustrates the basic flow of data through a Rete-based system.

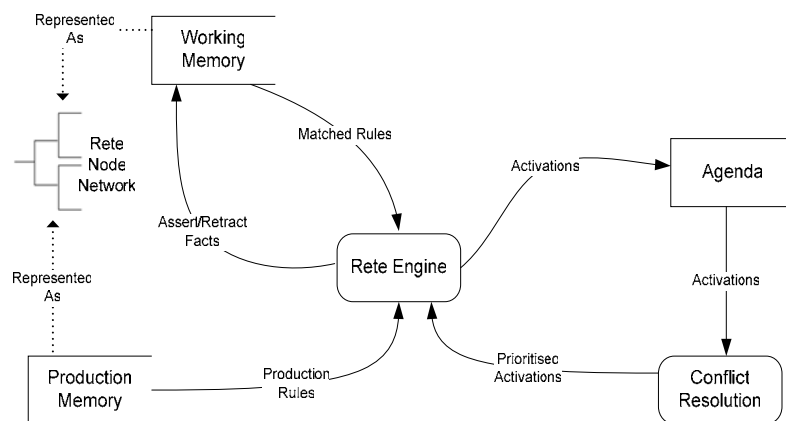


Figure 1 – Flow of data through a Rete-based systems

Prior to rule evaluation, the set of production rules will be compiled and translated into a Rete node network. Some nodes in this network represent the conditions in each of the rules, whilst other nodes in the network represent possible inter-relations between the conditions of each rule.





Tuples

A *tuple* is an ordered list of objects, each of a specified type. A tuple containing *n* objects is known as an *n-tuple*. A three-tuple is also known as a *triple*.

A tuple can be used to represent an object that is of a complex type. For instance, a *person* object could be represented by the tuple (Name, Age, Address).

The Rete engine is responsible for forming tuples from the facts in Working Memory.

Agenda

The *Agenda* maintains a collection of activations in prioritised queues. Execution of a set of rete rules involves the processing of all activations on the agenda until no further activations exist.

```
while Q.Count > 0
  Activation = Q.Pop
  Activation.Perform
end
```

To begin the evaluation process, the application asserts all known facts into the Rete which causes the Rete Algorithm to store these facts in working memory. These facts in working memory are known as *working memory elements*. The assertion of these facts will cause evaluation of rete network nodes that correspond to conditions that are dependant on the asserted facts (that is the USING clause for the rule contains a set of declarations that match a particular subset of the currently asserted facts). If the associated conditions evaluate to true, then a tuple containing the satisfying facts will be propagated to downstream network nodes. If the propagation of a tuple reaches the end of the network (a leaf node) then this indicates that all conditions for a specific production rule have been satisfied. The Rete engine will then place a reference to this rule, along with the final tuple that caused all the rules conditions to be satisfied, in another memory area known as the Agenda.

Strictly speaking, the Agenda is not native component of the Rete algorithm. However, most rules-based systems make use of an Agenda in order to manage the situation when multiple rules concurrently activate. The Agenda maintains a collection of *activations*, where an activation represents a rule, its actions, and the tuple of facts that caused the rule to activate. Once activations are on the Agenda, the action part of the rule may be executed and the activation is then removed from the Agenda. In the case where the Agenda contains more than a single activation, the Rete engine needs to determine the order in which to perform the actions associated with each activation. This may be further complicated if execution of an action causes new facts to be asserted, which may result in more activations being added to or removed from the Agenda. In this situation, a *conflict resolution strategy* is used to determine the order of execution when multiple activations are on the Agenda.

It should now be obvious that the sequence of execution of a Rete-based system can be very complex indeed. Additions of facts may result in actions being added to the agenda and executed, which may result in the addition of more facts. Additionally, facts may be retracted or modified at any stage during this process. Any changes to the facts may result in pending activations being removed from the Agenda before they are executed (if their conditions are no longer satisfied). This continual cycle of assertion-matching-execution will end only when there are no remaining activations on the Agenda.

Implementation of Rete in Common Knowledge

Common Knowledge includes a powerful Rete Rules Extender which allows the creation of production rules and their execution using an implementation of the Rete algorithm. The figure below shows the Common Knowledge Rete Rules editor.



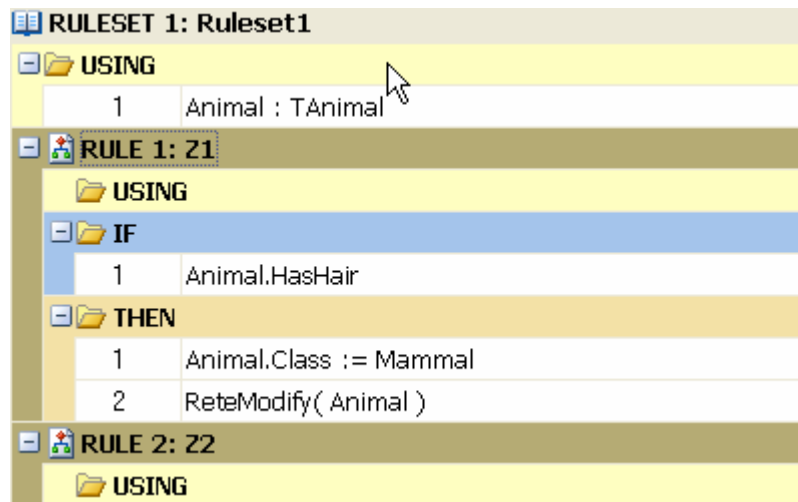


Figure 2 – Rules in Common Knowledge Rete Rules Editor

As well as allowing the creation of **USING-IF-THEN** styled production rules, Common Knowledge also allows rules to be grouped into *rulesets* which are logical grouping of rules. Associated with a ruleset is an additional set of **USING** declarations which allows you to define the fact classes that are relevant to all rules in the ruleset. Grouped together, these **USING** declarations form a tuple which is utilised by the Rete algorithm. If the facts currently in working memory can be combined to form a tuple that matches a set of **USING** declarations, then the corresponding rule-sets and contained rules will be evaluated or re-evaluated. Additionally, each individual rule is allowed to have its own set of **USING** declarations to allow the definition of a tuple relating to the specific rule only. Note that the tuple members of the ruleset are inherited by each rule in that ruleset. The Common Knowledge Rete Rules are made even more flexible by the fact that they can be combined with any of the other Common Knowledge Rule Extenders to create extremely powerful rule based solutions.

Execution of Rete Rules by the Common Knowledge Rules Engine is quick and efficient by virtue of a sophisticated implementation of the Rete algorithm. Fast execution is achieved through the construction of an efficient Rete node network and an efficient execution algorithm. The conflict resolution strategy can be controlled using rule priorities, a mechanism that allows rules to be assigned a priority of execution which is used to determine the order in which rule activations on the Agenda are performed.

Applications of Rete Rules

Rete Rules can be applied to a variety of problem types, where the total number of rules involved may be large, and it would be difficult to explicitly define the relationships between all permutations of the rules. Rete can be successfully applied to problems types including:

- Classification
- Pattern recognition
- Monitoring
- Planning
- Decision making



Rete Rules can be particularly effective where the problem requires an inferencing approach and state data needs to be maintained between rule executions. In these situations the data-driven approach taken by a Rete based engine is preferable to the procedural, statically-controlled approach that might be taken by a naive rules implementation.

An example of such an application might be a monitoring system that analyses signals produced by a complex system of alarms. Some alarms might signal frequently, some signal only occasionally, but all the alarms need to be considered when assessing whether some remedial action is required. A Rete approach utilising the Rete Rules Extender would be effective in this scenario as it would not need to continually re-evaluate the many slow-changing conditions. Rather, it would only re-evaluate conditions that are directly affected by the most recent change of facts.



Benefits of Common Knowledge Rete Rules

Rule Extenders

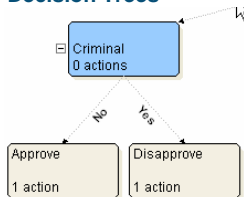
The set of Common Knowledge Rule Extenders includes the following (and more):

Decision Tables

	Rule1	Rule2	Rule3
Age	[0:2]	[0:2]	[3:8]
Weight	[0:8]	> 8	[0:20]
Dose	5	8	10

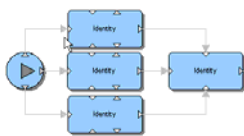
An intuitive, tabular format that provides a highly-maintainable and logical organisation of rules

Decision Trees



Useful for illustrating decision points and outcomes within a decision-making process.

Workflows



Allows business rules to be modelled within the flow of the larger business process.

Scripts

```

procedure main;
var
  Index: integer;
begin
  for Index := 1 to 8 do
  begin
    AnimalRules.Reset;
    AnimalRules.Execute();
  end;
end;
  
```

Procedural logic sometimes needs to be utilised when rule requirements do not easily match to one of the more standard rule formats.

The Common Knowledge Rete Rules Extender provides a powerful solution by utilising combining a production system rules format using the Rete algorithm with the additional rules capabilities of Common Knowledge. The benefits that can be gained by using the Rete Rules Extender to solve complex business problems may best be classified as follows:

- **Understandability** - the production system format of the Rete Rules Extender maps closely to the types of rules that a human would typically utilise whilst solving complex problems.
- **Readability** - rules are distilled into their most basic (declarative) form, allowing a divide-and-conquer approach to business rules when used in conjunction with other Rules Extender and Common Knowledge features.
- **Efficiency** - a large number of rules will quickly overwhelm attempts to build a naive or custom rules solution. Rete will provide guaranteed efficiency when there are a large number of rules.
- **Expressiveness** - the flexible nature of the production rules format, combined with the power of Common Knowledge expressions, provides a rich vocabulary with which to represent the conditions and actions associated with a rule.

The Rete Rules Extender can be easily combined with other Common Knowledge Rules Extenders such as Decision Trees, Decision Tables, Decision Grids, Workflows and Scripts. For example, a set of Rete Rules could be seamlessly integrated as a classification step of a Workflow or, perhaps, a Decision Table could be used to evaluate a particular condition of a rule within a Rete Ruleset. Common Knowledge Expressions are the enabler that facilitates this integration. Expressions combine terms and operators where the terms may be as low level as a simple data value or as complex as the evaluation of an entire ruleset. Expressions are used in both the IF portion and the THEN portion of any individual Rete rule, meaning that any Rules Extender can be incorporated into both the conditions and actions of a Rete rule. This flexibility allows a rules designer to pick and choose the most appropriate format when constructing rules. For example, the designer may choose a Decision Table to provide a highly readable representation of a particular subset of rules, but may choose to use the Rete Rules Extender for the remaining rules in order to leverage the fast execution of the Rete algorithm.



Summary

••• The Rete algorithm is an efficient and powerful algorithm for the execution of a large number of rules within a rules-based system. Common Knowledge utilises this widely-accepted algorithm within its Rete Rules Extender. Common Knowledge Studio provides the ability to combine the flexibility of the production system rules format with the power of Common Knowledge's expressions and the other Rule Extenders. The Common Knowledge Rules Engine provides for fast, efficient and scalable execution of these rules by its use of the Rete algorithm. This combined power allows for the building of rules solutions for a large number of problem types whilst benefiting from the intuitive and expressive nature of this rule format.

Sydney
Suite 4, Ground Floor, 13a Narabang Way, Belrose NSW 2085, AUSTRALIA
T. +61 2 9450 2999 F. +61 2 9450 2744

Newcastle
Unit 2, 3 Bradford Close, Kotara NSW 2289, AUSTRALIA
T. +61 2 4957 3146

W. www.objectconnections.com E. info@objectconnections.com

