Different kinds of rules and how to write them properly

> *This column is the next in a series that provides the reader with best practices on using or choosing a rules engine. The target audience for this series is typically the user of a rule engine, i.e., a programmer or someone with programming skills. All coding examples should be read as pseudo-codeand should be easily translated to a specific target syntax for a rule engine that supports backward and forward chaining in an object-oriented environment.*

We will discuss recommendations on the question of which different kinds of rules and inferencing methods are (potentially) available for rule engine users. In this description the following concepts are important:

- Attribute: symbolic representation[1] used to denote a quantity or expression
- Premise: a set of conditions connected with a Boolean logical operator
- Action: one or more conclusions
- Condition: a programming construct that evaluates to true or false
- Conclusion: a programming construct that sets a value for an attribute or calls a method
- Rule: a programming construct that is used to describe the relationship between attributes in your domain model
  A rule can be a statement, which has to be true at all time, but it can also be a description of how an attribute value can be derived from other attributes' values.
- Inference engine: the mechanism that is able to solve the goal of your inference task with help of the provided rules

- Rule service: the collection of inferences, rule sets, and rules that make up a service

A production rule is a programming construct that is used to describe the relationship between attributes in your domain model. Production rules consist of one or more conditions that are combined with a logical operator (conjunction or disjunction) and one or more conclusions. Some rule engines only allow only one conclusion. Once a rule's condition evaluates to true the rule conclusion is executed. The conclusion can be an attribute value assignment; it can also be a method-call.

Most inference engines support different rule types with slightly different semantics. Here is an overview.

Kinds of rule types

If-rules: In a backward chaining process the inference engine will 'enter' the rules through their actions (conclusion), while in a forward chaining process the rules are 'entered' through their premise (conditions). Once the conclusion of a rule is reached, the rule will not be re-evaluated in the same session.

When-rules: When-rules are sometimes referred to as *daemons*, which are monitoring rules; they will be (re)evaluated whenever the state of their premise is affected. In other words, whenever an attribute mentioned in the premise of a daemon is changed, the rule is (re)evaluated.

Decision tables: The decision table is a special construct that is able to bundle several if-rules in a compound 'statement'. Its advantage over if-rules is its readability. I often consider a decision table to be a rule set — it's a set of rules with similar premises and similar actions. Inference engines are often optimized to take the shortest route through the decision table; once the inference engine has reached a conclusion it will stop evaluating other rows.

Kinds of reasoning semantics

Backward chaining: a recursive algorithm for executing production rules
Also known as *goal-driven reasoning*, backward chaining seeks to establish a value of an attribute (or "goal") by ascertaining the truth of the conditions of production rules whose action assigns a value to the attribute. Unknown attributes in those conditions are considered subgoals and are similarly pursued.

Forward chaining: a class of algorithms for executing production rules
Also, known as *data-driven reasoning*, forward chaining executes production rules by testing whether the rule's condition is true. Simple forward chaining is used to assign attribute values based on other attribute values. More complex forward chaining algorithms support first-order predicate calculus, i.e., quantification over instances of classes, and are executed by means of the Rete algorithm.

Declarative logic

If-rules are the declarative counterpart of the procedural if…then…else(if)… programming construct. The difference between the two is that in the case of if-rules the inference engine will

decide in what order the rules are evaluated. If the rules are consistent, complete, and non-redundant one should not worry about the order of the rule statements.

When to use backward chaining

Backward chaining should be used when the rules are used to guide the user through a dialog that gathers the required information when reasoning to conclude a goal that is known in advance. With backward chaining only the rules that are relevant to the specific case will be evaluated and information that is not required to make a conclusion is not derived.

When to use forward chaining

Forward chaining should be used when multiple goals (attributes) can be concluded and it is not known in advance which of the goals must be concluded to be successful. Also, forward chaining can be used when it is desirable to derive as much information as possible.

Constraints

If you want to define a relationship between attributes that has to be true at all times — e.g., the truth of this statement has to be 'guarded' — you use a when-rule.

The premise of these daemons expresses the relationship, and the action establishes an exception. Or, in rare occasions you may want to use the action to reestablish the truth.

example when-rule

```
when age < 18
and application accepted
then
raise an error
end
when age < 18
then
application is not accepted
end
```

In this way, whenever there exists an undesired condition in your domain, you can take the appropriate actions. I'd like to refer to my definition of declarative rule here: The daemons tend to express instructions, rather than statements.

Decision tables

Decision tables can be used when you have a lot of rules that use the same attributes in the condition and conclusion parts of the rule. The table really represents a set of rules where the rule template for all rules is the same and the rules only differ in the values they test an attribute for.

example decision table

Gender	Age	Risk
male	<= 30	High
male	> 30	High
female	<= 20	High
female	> 20	Low

Support by different vendors is differentiated based on what is supported as an acceptable value for the table cells. Most decision table support systems only allow a test on a numerical value or literal in the cells. However, some vendors support the test on an attribute (variable) in the cell of a decision table. The number of action columns can be more than 1 in some environments, and one can often execute a function in the action column.

Did you learn something new in reading this article? Let me know by sharing!

This article was originally published by BRCommunity (link).

References

[1] See ‘symbol’: http://en.wikipedia.org/wiki/Symbol