Rules

We have facts....
Let's use them with **Rules**
In an Expert system, the rules are of the form:

If *certain conditions exists*
Then *do some actions*

An Example

Rule:
If animal is a duck
Then the sound that it makes is quack
In CLIPS:

```
CLIPS> (defrule duck
   "Here comes the quack"
   (animal-is duck)
=>
   (assert (sound-is quack)))
CLIPS>
```

The pattern **(animal-is duck)** will matched against the facts. If it exists, then the **(sound-is quack)** fact will be asserted

General Form: defrule

```
(defrule rule_name "Not so optional description of function"
     (pattern_1)      ; The Left-Hand-Side LHS
     (pattern_2)
          .
          .
          .
     (pattern_N)
=>
     (action_1)       ; The Right-Hand-Side RHS
     (action_2)
          .
          .
          .
     (action_M)
               )      ; The ending parenthesis
```

Patterns and Actions

Patterns These are matched against the current facts and if the *all* match, then the actions are taken. There can be zero or more patterns.

Actions These are what are performed when the patterns are matched. They are usually are commands such as **assert**, **retract** or maybe some printout

Execution

Execution of a rule system
is the matching of patterns
within the rules.

When no pattern can be matched
Execution Stops

Agenda

```
CLIPS>(assert (sound-is quack))
<Fact-0>
CLIPS> (defrule duck
            (animal-is duck)
        =>
            (assert (sound-is quack)))
CLIPS> (agenda)
0      duck: f-0

For a total of 1 activation.
```

Executing

```
CLIPS> (facts)
f-0     (animal-is duck) CF 1.00

For a total of 1 fact.

CLIPS> (run)
==> f-1     (sound-is quack) CF 1.00

CLIPS> (facts)
f-0     (animal-is duck) CF 1.00

f-1     (sound-is quack) CF 1.00

For a total of 2 facts.

CLIPS> (agenda)
CLIPS> (run)
CLIPS>
```

Notice when run again, nothing happens. Notice also that the **agenda** is empty.

When is a Rule Activated

- A brand new pattern entity that did not exist before
- a pattern entity that did exist before, but was retracted and reasserted

Sorting of Rules

The Inference Engine sorts the activations according to their
Salience

As seen by the **agenda** command

After firing, the rule is eliminated from the agenda

Without this, the engine would be caught in infinite loops

An Explicite Example

```
CLIPS> (clear)
CLIPS> (defrule duck "The quack rule"
(animal-is duck)

=>

(printout t "quack" crlf))

CLIPS> (watch facts)
CLIPS> (watch activations)

CLIPS> (assert (animal-is duck))
==> f-0     (animal-is duck) CF 1.00

==> Activation 0      duck: f-0

<Fact-0>

CLIPS> (assert (animal-is duck))
FALSE


CLIPS> (agenda)
0      duck: f-0

For a total of 1 activation.
```

```
CLIPS> (run)
quack


CLIPS> (agenda)

CLIPS> (facts)
f-0     (animal-is duck) CF 1.00

For a total of 1 fact.

CLIPS> (run)
CLIPS>
```

Looking at Rules

```
CLIPS> (rules)
duck

For a total of 1 defrule.



CLIPS> (ppdefrule duck)
(defrule MAIN::duck "The quack rule"

   (animal-is duck)

   =>

   (printout t "quack" crlf))
```

Forward Chaining

Expert Systems
Rule Execution through Forward Chaining
(Activation by RHS) PROLOG

Rule Execution through Backward Chaining
(Activation by LHS)