Jess:
A Production System Language

4.209 Agent-Based Virtual Worlds

Mary Lou Maher MIT Fall 2002

Jess Knowledge Base

- A rule-based system maintains a collection of knowledge nuggets called *facts*. This collection is known as the *knowledge base*. It is somewhat akin to a relational database, especially in that the facts must have a specific structure.
- In Jess, there are three kinds of facts:
  - *ordered facts*,
  - *unordered facts*, and
  - *definstance facts*.

Ordered Facts

- Ordered facts are simply lists, where the first field (the *head* of the list) acts as a sort of category for the fact. Here are some examples of ordered facts:

- `(shopping-list eggs milk bread)`
- `(person "Bob Smith" Male 35)`
- `(father-of danielle ejfried)`

Assert Function

```
Jess> (reset)
  TRUE
  Jess> (assert (father-of danielle ejfried))
  <Fact-1>
  Jess> (facts)
  f-0   (MAIN::initial-fact)
  f-1   (MAIN::father-of danielle ejfried)
  For a total of 2 facts.
```

Unordered Facts

Ordered facts are useful, but they are unstructured. Sometimes (most of the time) you need a bit more organization. In object-oriented languages, *objects* have named *fields* in which data appears. Unordered facts offer this capability (although the fields are traditionally called *slots*.)

```
 (person (name "Bob Smith") (age 34) (gender
Male))
 (automobile (make Ford) (model Explorer) (year
1999))
```

Deftemplate

- Unordered facts cannot be asserted until you define their structure:
- (deftemplate <deftemplate-name> [extends <classname>] [<doc-comment>]
- [(slot <slot-name> [(default | default-dynamic <value>)]
- [(type <typespec>)]*)

- There may be an arbitrary number of slots. Each <slot-name> must be an atom. The default slot qualifier states that the default value of a slot in a new fact is given by <value>; the default is the atom nil. The 'default-dynamic' version will evaluate the given value each time a new fact using this template is asserted. The 'type' slot qualifier is accepted but not currently enforced by Jess; it specifies what data type the slot is allowed to hold. Acceptable values are ANY, INTEGER, FLOAT, NUMBER, ATOM, STRING, LEXEME, and OBJECT.

Jess Rulebase

```
Jess> (defrule do-change-baby
   "If baby is wet, change baby's
 diaper."
   (baby-is-wet)
   =>
   (change-baby))
```

Wrong Rule

Left hand side of rule matches facts, not functions:

```
Jess> (defrule wrong-rule
      (eq 1 1)
      =>
      (printout t "Just as I thought,  1
 == 1!" crlf))
```

Matching Patterns

```
Jess> (defrule example-2
    (a ?x ?y)
    =>
    (printout t "Saw 'a " ?x " " ?y "'" crlf))

Jess> (defrule example-3
    (not-b-and-c ?n1&~b ?n2&~c)
    (different ?d1 ?d2&~?d1)
    (same ?s ?s)
    (more-than-one-hundred ?m&:(> ?m 100))
    (red-or-blue red|blue)
    =>
    (printout t "Found what I wanted!" crlf))

(deftemplate person (slot age))
  Jess> (defrule example-8
     (person (age ?x))
     (test (> ?x 30))
     =>
     (printout t ?x " is over 30!" crlf))
```

Modules

A module defines a *namespace* for templates and rules. This means that two different modules can each contain a rule with a given name without conflicting -- i.e., rules named `MAIN::initialize` and `COMMUTE::initialize` could be defined simultaneously and coexist in the same program. Similarly, the templates `COMPUTER::bus` and `COMMUTE::bus` could both be defined. Given this fact, there is the question of how Jess decides which template the definition of a rule or query is referring to.

Compiling a rule

When Jess is compiling a rule or deffacts definition, it will look for templates in three places, in order:

1. If a pattern explicitly names a module, only that module is searched.
2. If the pattern does not specify a module, then the module in which the rule is defined is searched first.
3. If the template is not found in the rule's module, the module MAIN is searched last. Note that this makes the MAIN module a sort of global namespace for templates.

Example

- Monkeys and bananas

Salience

Each rule has a property called *salience* that is a kind of rule priority. Activated rules of the highest salience will fire first, followed by rules of lower salience. To force certain rules to always fire first or last, rules can include a salience declaration:

```
Jess> (defrule example-6
  (declare (salience -100))
  (command exit-when-idle)
  =>
  (printout t "exiting..." crlf))
```

The use of salience is discouraged in rule-based programming

Deftemplate example

```
Jess> (deftemplate automobile
   "A specific car."
   (slot make)
   (slot model)
   (slot year (type INTEGER))
   (slot color (default white)))

Jess> (assert (automobile (make Chrysler) (model
  LeBaron)(year 1997)))
  <Fact-0>
Jess> (facts)
  f-0   (MAIN::automobile (make Chrysler) (model
  LeBaron)(year 1997) (color white))
```