The Java Expert System Shell

Martin Strauss, 26.10.2006
All pants different colours
One is wearing red pants
The golfer to Fred’s immediate right is wearing blue pants
Joe is second in line
Bob is wearing plaid pants
Tom isn’t in position 1 or 4
Tom isn’t wearing the hideous orange pants
All pants different colours
One is wearing red pants
The golfer to Fred’s immediate right is wearing blue pants
Joe is second in line
Bob is wearing plaid pants
Tom isn’t in position 1
Tom isn’t wearing the hideous orange pants
Expert Systems
Expert Systems

- \( \forall w, x, y, z \text{ pants}(w, x) \land \text{ pants}(y, z) \land (z \neq x \lor y = w) \)
- \( \exists x \ \text{ pants}(x, \text{ red}) \)
- \( \exists x, y \ \text{ position}(\text{Fred}, x) \land \text{ position}(y, x+1) \land \text{ pants}(y, \text{ blue}) \)
- \( \text{ position}(\text{Joe}, 2) \)
- \( \text{ pants}(\text{Bob}, \text{ plaid}) \)
- \( \exists x \ \text{ position}(\text{Tom}, x) \land x \neq 1 \land x \neq 4 \)
- \( \exists x \ \text{ pants}(\text{Tom}, x) \land x \neq \text{ orange} \)
Expert Systems

- \( \forall w, x, y, z \ \text{pants}(w, x) \land \text{pants}(y, z) \land (z \neq x \lor y = w) \)
- \( \exists x \ \text{pants}(x, \text{red}) \)
- \( \exists x, y \ \text{position}(Fred, x) \land \text{position}(y, x + 1) \land \text{pants}(y, \text{blue}) \)
- \( \text{position}(Joe, 2) \)
- \( \text{pants}(Bob, \text{plaid}) \)
- \( \exists x \ \text{position}(Tom, x) \land x \neq 1 \land x \neq 4 \)
- \( \exists x \ \text{pants}(Tom, x) \land x \neq \text{orange} \)

- \( \forall x \ \text{position}(x, ?) \)
- \( \forall x \ \text{pants}(x, ?) \)
Expert Systems

- $\forall w, x, y, z \text{ pants}(w, x) \land \text{ pants}(y, z) \land (z \neq x \lor y = w)$
- $\exists x \text{ pants}(x, \text{ red})$
- $\exists x, y \text{ position}(Fred, x) \land \text{ position}(y, x + 1) \land \text{ pants}(y, \text{ blue})$
- $\text{ position}(Joe, 2)$
- $\text{ pants}(Bob, \text{ plaid})$
- $\exists x \text{ position}(Tom, x) \land x \neq 1 \land x \neq 4$
- $\exists x \text{ pants}(Tom, x) \land x \neq \text{ orange}$

- $\text{ position}(Fred, 1)$
- $\text{ pants}(Fred, \text{ orange})$
- $\text{ position}(Joe, 2)$
- $\text{ pants}(Joe, \text{ blue})$
- $\text{ position}(Bob, 3)$
- $\text{ pants}(Bob, \text{ plaid})$
- $\text{ position}(Tom, 4)$
- $\text{ pants}(Tom, \text{ red})$
Declarative programming

Specify *what* not *how*

Specifically for logic, we specify:

- facts
- rules
Declarative programming

- Specify *what* not *how*
- Specifically for logic, we specify:
  - facts
  - rules

Rule-based system
Rule-based system

Inference engine

Pattern matcher

Agenda

Execution engine

Working memory

\[ \exists x \text{ pants}(x, \text{red}) \]

\[ \exists x, y \text{ position}(\text{Fred, } x) \land \text{position}(y, x + 1) \land \text{pants}(y, \text{blue}) \]

etc...

Rule base

\[ \exists x, y \text{ pants}(x, y) \rightarrow \text{print}(x \text{ " wears " } y) \]

\[ \exists x, y \text{ position}(x, y) \rightarrow \text{print}(x \text{ " is at position " } y) \]

etc...
Introducing JESS

Inference engine

Pattern matcher

Agenda

Execution engine

Java

Working memory

(defrule form-1040EZ
  (user (income ...))
  =>
  (assert (recommendation (form 1040EZ))
  ...

pants(x, red)

position(Fred, x)

position(y, x + 1)

pants(y, blue)

e tc ...

x, y  pants(x, y) ➝ print(x " wears " y)

x, y  position(x, y) ➝ print(x " is at position " y)

e tc...
The JESS language

- LISP-like syntax
  - (this (means that) everything (looks like a) (list))

(pants-colour (of Fred) (is ?c1))
(user (income ?i) (dependents ?d))

- Whitespace is ignored
The JESS language

Symbols

[a-zA-Z0-9$*.=+/<>_?#]+

first_value
contestant#1
_abc
_ABC

1st_value
?question
$lots
=abc

nil
TRUE
FALSE
crlf
The JESS language

- **Symbols**
  - `[a-zA-Z0-9$*.=+/<>_?#]+`

```
first_value  1st_value  nil
contestant#1  ?question  TRUE
_abc         $lots      FALSE
_ABC         =abc       crlf
```
The JESS language

- **Values**
- **Symbols** filing-status
- **Numbers** 65,000
- **Strings** “what is your income?”
- **Comments** ; this is a comment
The JESS language

- Variables
  - Standard
    - ?income
  - Global
    - ?*name*
    - (defglobal ?*x* = 3)
  - Multifield
    - $?forms

- Assigning:
  - (bind ?expl "Expenses")

Untyped!
The JESS language

- **Structures**
  - **Lists**
    - Handling lists: create$, nth$, first$, rest$
    - Functions: (+ 2 3)
    - Define: deffunction
  - (1040 1040EZ 2411)
The JESS Language

- Useful functions
  - (facts)
  - (rules)
  - (watch facts)
The JESS language

Defining a function:

(deffunction is-of-type (?answer ?type)
  "Check that the answer has the right form"
  (if (eq ?type yes-no) then
    (return (or (eq ?answer yes) (eq ?answer no)))
  else (if (eq ?type number) then
           (return (numberp ?answer)))
  else (return (> (str-length ?answer) 0)))))
The JESS language

- **Facts**

  **Ordered**
  
  (answer income 45000)

  ![we have an answer about income, it’s 45000](image)

  **Unordered**
  
  (answer (ident income)
  (text 45000))

  ![we have an answer ident = income text = 45000](image)
The JESS language

- **Facts**

  **Ordered**
  
  (ask childcare)

  **Unordered**
  
  (answer (ident income) (text 45000))

we need to ask about childcare

we have an answer
  ident = income
text = 45000
The JESS language

Facts

Ordered
(ask childcare)

Unordered
(ask childcare)
(we have an) Answer
(ident income)
(text 45000)

(ask childcare)
(we need to ask about childcare)

(deftemplate answer
"answer to a question"
(slot ident)
(slot text))
The JESS language

Facts

Ordered
(ask childcare)

Unordered
(answer (ident income)
  (text 45000))

we need to ask about childcare

(we have an) Answer
(ident = income)
(slot text = "no")
Facts

Working memory

\[ \exists x, y \ \text{position}(Fred, x) \land \text{position}(y, x + 1) \land \text{pants}(y, \text{blue}) \]

etc...

Rule base

\[ \exists x, y \ \text{pants}(x, y) \rightarrow \text{print}(x \ " \text{wears} \" \ y) \]
\[ \exists x, y \ \text{position}(x, y) \rightarrow \text{print}(x \ " \text{is at position} \" \ y) \]

etc...
Facts

Working memory

\[ \exists x \ \text{pants}(x, \text{red}) \]
\[ \exists x, y \ \text{position}(\text{Fred}, x) \land \text{position}(y, x + 1) \land \text{pants}(y, \text{blue}) \]
\[ \text{etc...} \]

Rule base

\[ \exists x, y \ \text{pants}(x, y) \rightarrow \text{print}(x \ "\text{wears}\" y) \]
\[ \exists x, y \ \text{position}(x, y) \rightarrow \text{print}(x \ "\text{is at}\" \ y) \]
\[ \text{etc...} \]
Facts

Working memory

\[ \exists x \, \text{pants}(x, \text{red}) \]

etc...

Rule base

\[ \exists x, y \, \text{pants}(x, y) \rightarrow \text{print}(x \, \text{"wears"} \, y) \]

\[ \exists x, y \, \text{position}(x, y) \rightarrow \text{print}(x \, \text{"is at position"} \, y) \]

etc...
Facts

- Inference engine
- Execution engine
- Working memory
  - Rule base
    - $\exists x \ pants(x, \text{red})$
    - $\exists x, y \ pants(x, y) \rightarrow \text{print}(x \ " \text{wears} \ " \ y)$
    - $\exists x, y \ position(x, y) \rightarrow \text{print}(x \ " \text{is at position} \ " \ y)$
    - etc...
- Pattern matcher
- Agenda

**assert**

**retract**

**clear**
Facts

Working memory

\[ \exists x \text{ pants}(x, \text{red}) \]
\[ \text{etc...} \]

Rule base

\[ \exists x, y \text{ pants}(x, y) \rightarrow \text{print}(x \text{ " wears " } y) \]
\[ \exists x, y \text{ position}(x, y) \rightarrow \text{print}(x \text{ " is at position " } y) \]
\[ \text{etc...} \]
Facts

- Inference engine
- Execution engine
- Working memory
- Rule base

**Pattern matcher**

- assert
- retract

**Agenda**

- clear
- reset
- initial-fact

**Working memory**

\[ \exists x \text{ pants}(x, \text{red}) \]

etc...

**Rule base**

\[ \exists x, y \text{ pants}(x, y) \rightarrow \text{print}(x \text{ “ wears ” } y) \]

\[ \exists x, y \text{ position}(x, y) \rightarrow \text{print}(x \text{ “ is at position ” } y) \]

etc...
Facts

- Working memory
  - \( \exists x \text{ pants}(x, \text{red}) \)
  - etc...
- Rule base
  - \( \exists x, y \text{ pants}(x, y) \rightarrow \text{print}(x \text{ " wears " } y) \)
  - \( \exists x, y \text{ position}(x, y) \rightarrow \text{print}(x \text{ " is at position " } y) \)
  - etc...

- assert
- retract
- clear
- reset
- initial-fact
- deffacts
The JESS language

- **Conditional elements**
  - (and (a) (b)) \( a \land b \)
  - (or (a) (b)) \( a \lor b \)
  - (not (a)) \( \neg a \)
  - (exists (a)) \( \exists a \)
The JESS language

- Conditional elements
  - `(test (< (?a) (?b)))`

- logical
The JESS language

Rules

(defrule form-2441
(answer (ident childcare) (text yes))
=>
(assert (recommendation (form 2441)
(explanation "Child care expenses")))))
Rules

- **Variables in slot values**

  \[(\text{user} \ (\text{income} \ ?i) \ (\text{dependents} \ ?d)) \Rightarrow \ldots\]
  
  \(?\text{ask} \leftarrow (\text{MAIN}::\text{ask} \ ?id) \Rightarrow \ldots\)

- **Constraints**

  \[(\text{user} \ (\text{income} \ ?i&\:(< \ ?i \ 50000)))
    \quad (\text{dependents} \ ?d&\:(\text{eq} \ ?d \ 0))) \Rightarrow \ldots\]
What about logical?

(defrule form-2441
  (answer (ident childcare) (text yes)))

=>

(assert (recommendation (form 2441)
  (explanation "Child care expenses")))
What about logical?

(defrule form-2441

  (logical (answer (ident childcare) (text yes)))

  =>

  (assert (recommendation (form 2441)
    (explanation "Child care expenses"))))
Rule execution

**Rule base**

(pants (of ?x) (colour ?y)) → (printout t ?x "wears" ?y)
(position (of ?x) (is ?y)) → (printout t ?x "is at position" ?y)

**Working memory**

(pants (of ?x) (colour red))
(position (of Fred) (is ?x)) and
(position (of ?y) (is ?x + 1)) and
(pants (of ?y) (colour blue))

**Inference engine**

**Pattern matcher**

**Execution engine**

**Agenda**
Rule execution

Inference engine

Pattern matcher

Agenda

Execution engine

Working memory

(pants (of ?x) (colour red))

(position (of Fred) (is ?x)) and
(position (of ?y) (is ?x + 1)) and
(pants (of ?y) (colour blue))

Rule base

(pants (of ?x) (colour ?y)) → (printout t ?x "wears" ?y)

(position (of ?x) (is ?y))

→ (printout t ?x "is at position" ?y)
Rule execution

Rule base

**module ask**

(ask ?x)

(etc...)

**module recommend**

(module recommend

(answer (ident childcare) (text "yes") => (recommend (form 2441) (explanation "Child care expenses")))

(etc...)
(defmodule interview)
(focus startup interview recommend report)
(clear-focus-stack)
(pop-focus)
(get-current-module)
(declare auto-focus TRUE) & (return)
Rule execution

- `(defmodule interview)`
- `(focus startup interview recommend report)`
- `(clear-focus-stack)`
- `(pop-focus)`
- `(get-current-module)`
- `(declare auto-focus TRUE) & (return)`
Rule execution

- Name resolution
  - ask::ask-question-by-id
  - is-of-type
  - MAIN::question
The JESS language

Control flow

- (foreach ?e ?golfers-list
  (printout t ?e crlf))
- (while (not (is-of-type ?answer ?type)) do
  (printout t ?question crlf))
- (if (eq ?type yes-no) then
  (printout t "(yes or no)") else
  (printout t "(numeric)")
The JESS language

Control flow

- (foreach ?e ?golfers-list
   (printout t ?e crlf))

- (while (not (is-of-type ?answer ?type)) do
   (printout t ?question crlf))

- (if (eq ?type yes-no) then
   (printout t "(yes or no)")) else
   (printout t "(numeric)")

Not declarative
The JESS language

- Control flow

  - (while (progn (bind ?n (* ?n ?n) (< ?n 1000) do
      (printout t ?n crlf))
  - (apply ?function ?x ?y)
  - (eval "(+ 1 2 3")
  - (build "(+ 1 2 3")
An example application

What is your income?

$45,000

You need form 1040A

OK
The knowledge base

- #1040 is the standard long form.
- #1040A is the short form
  - use instead of 1040 if income < $50,000
  - no deductions
- #1040EZ is the very short form
  - use instead of 1040A if income < $50,000, no dependents, no deductions, no taxable interest > $400.
- #2441 is for daycare expenses (including elderly parents and other dependents)
- #2016EZ is for unreimbursed work expenses (primarily travel)
- #3903 is for unreimbursed moving expenses if you moved (further than 50 miles) because of your job
- #4684 is for recovering losses
- #4868 is for applying for an extension for filling out taxes
- #8283 is for credit for donating more than $500 to charity
- #8829 is for deducting home office expenses
(defrule form-1040EZ
  (user (income ?i&:(< ?i 50000))
        (dependents ?d&:(eq ?d 0)))
  (answer (ident interest) (text no))
  =>
  (assert (recommendation
    (form 1040EZ)
    (explanation "Income below threshold, no dependents"))))
Infrastructure

startup -> interview -> recommend -> report

ask
Infrastructure

(startup -> interview)

(ask -> interview)

(defrule form-1040EZ
  (user
    (income ...))
  =>
  (assert
    (recommendation
      (form 1040EZ)
    ...

(report -> interview)
Infrastructure

(defrule print-banner =>
 (printout t crlf "***********
 crlf)
 (printout t " Hello "))...

(defrule form-1040EZ (user (income ...)) =>
 (assert (recommendation (form 1040EZ) ...

ask interview report
(defrule print-banner =>
  (printout t crlf "***********" crlf)
  (printout t "Hello "))...

(defrule form-1040EZ (user (income ...)) =>
  (assert (recommendation (form 1040EZ) ...)
    (printout t "*** Please take..."))
)

The Questions

(deffacts question-data
  "The questions the system can ask."
  (question (ident income) (type number)
    (text "What was your annual income?"))
  (question (ident interest) (type yes-no)
    (text "Did you earn more than $400 of taxable interest?"))
  (question (ident dependents) (type number)
    (text "How many dependents live with you?"))
  ...
)
The “ask” module

(defrule ask::ask-question-by-id
  "Given the identifier of a question, ask it and assert the answer"
  (declare (auto-focus TRUE))
  ?ask <- (MAIN::ask ?id)
  (MAIN::question (ident ?id) (text ?text) (type ?type))
  (not (MAIN::answer (ident ?id)))
  =>
  (bind ?answer (ask-user ?text ?type))
  (assert (answer (ident ?id) (text ?answer)))
  (retract ?ask)
  (return))
(defrule request-income
=>
(assert (ask income)))
(defrule assert-user-fact
(answer (ident income) (text ?i))
(answer (ident dependents) (text ?d))
=>
(assert (user (income ?i) (dependents ?d))))
The "interview" module

(defrule request-childcare-expenses
  ;; If the user has dependents
  (answer (ident dependents) (text ?t &:(> ?t 0)))
  =>
  (assert (ask childcare)))
The Application

(defrule print-banner
  =>
  (printout t
crlf
"**********"crlf)
  (printout t "Hello "))...

(defrule request-income
  =>
  (assert (ask income)))
(defrule request-num-dependents
  => ...

(defrule ask-question-by-id
  (MAIN::question
  (ident ?id)
  (text ?text)
  (type ?type))
  ...

(defrule sort-and-print
  ?r1 <-
  (recommendation
  (form ?f1)
  (explanation ?
e)) ...
  =>
  (printout t
  "*** Please take
  (defrule form-1040EZ
  (user
  (income ...))
  =>
  (assert
  (recommendation
   (form 1040EZ)
  (defrule print-banner
  =>
  (printout t
crlf
"**********"crlf)
  (printout t "Hello "))...
The Application

(defrule sort-and-print
  ?r1 <-
  (recommendation
    (form ?f1)
    (explanation ?e))
  =>
  (printout t
    "*** Please take
    ***
    (defrule form-1040EZ
      (user
        (income ...))
      =>
      (assert
        (recommendation
          (form 1040EZ)
          ...
          (defrule ask-question-by-id
            (MAIN::question
              (ident ?id)
              (text ?text)
              (type ?type))
            ...
            (defrule request-income
              =>
              (assert (ask income)))
            (defrule request-num-dependents
              => ...

(deffunction run-system ()
  (reset)
  (focus startup interview recommend report)
  (run))

(while TRUE
  (run-system))