Artificial Intelligence

Lab 1

Marco Piastra

Jess?

Acronym of *Java Expert System Shell*

- A small environment written in Java

  Author: Ernest Friedman-Hill, Sandia National Laboratories in Livermore, Canada

  Rule-based system: the main programming construct is the *rule*

  if <cond> then <action>

  It is derived from an older and much larger system: CLIPS

  Adopts the syntax of the LISP programming language

  A program in Jess is mostly made up of a set *rules*
  to be applied to a set of *facts*

Java: using the `CLASSPATH` variable

You do not know what this is?
It's normal: you never used Java at all

Impostazione della variabile `CLASSPATH`

```
$ CLASSPATH=<value>
$ export CLASSPATH
```

`$` is the system prompt (do not write it, the system does)

To make sure that the above worked

```
$ echo $CLASSPATH
```

In our case:

```
$ CLASSPATH=/home/opt/Jess61p8
$ export CLASSPATH
```

Make sure you respect small and capital letters! (It's Linux, not Windows)

Starting Jess

```
$ java jess.Main
Jess>
```

Quitting Jess

```
Jess> (exit)
$
```

(Wow, you made it!)

Lists

(file finitestatemachine.jess)

- The LISP syntax

  It could hardly be simpler: everything is either an symbol or a list.

  LISP: Lots of Impossible Stupid Parenthesis

  Example:

```
(deftemplate fsm
        (slot current-state)
        (multislot input-stream)
        (multislot output-stream)
)
```

  Lists can be nested at will (make sure you balance parentheses)

  Example:

```
(exit)
```

  Every Jess command is a list.

- *Every legitimate Jess expression is a list*

deftemplate

(file finitestatemachine.jess)

- It defines *template* for structured *facts* (=basic data items in Jess)

  Example:

```
(deftemplate event
        (slot current-state)
        (slot input-symbol)
        (slot output-symbol)
        (slot new-state)
)
```

  It is the template of individual facts like this:

```
(event
        (current-state even)
        (input-symbol 1)
        (output-symbol 1)
        (new-state odd)
)
```

  Every slot will have exactly one value

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

The meaning: the whole Jess language could fit it in.

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

The whole thing is a list

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example: Name of the rule, just an id

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
    =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

Separator: just find this first

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

(*LHS - Left Hand Side*): the logical conditions that determine the applicability of the rule

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
     ?current <- (fsm (current-state ?cs)
                      (input-stream ?is $?rest)
                      (output-stream $?output))
     (event (current-state ?cs)
            (input-symbol ?is)
            (output-symbol ?os)
            (new-state ?ns))
      =>
     (printout t "From state " ?cs " input " ?is
                 " to state " ?ns " output " ?os crlf)
     (modify ?current (current-state ?ns)
                      (input-stream ?rest)
                      (output-stream ?os ?output))
)
```

Think about the LHS as the description of a *pattern* to be applied to facts

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

(*RHS - Right Hand Side*): it describes the actions to be performed when the rule is *FIRE*d

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

Facts can be either *asserted* or *retracted*

Messages can be printed

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

This is a variable

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

A Jess variable
is always preceded by ?

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

If the overall pattern matches each variable will bind to a symbol

CAUTION:
matching the pattern in the LHS is an "all or nothing" matter…

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

A variable can also bind to a fact

```
(defrule state-transition
   ?current <- (fsm (current-state ?cs)
                    (input-stream ?is $?rest)
                    (output-stream $?output))
   (event (current-state ?cs)
          (input-symbol ?is)
          (output-symbol ?os)
          (new-state ?ns))
    =>
   (printout t "From state " ?cs " input " ?is
               " to state " ?ns " output " ?os crlf)
   (modify ?current (current-state ?ns)
                    (input-stream ?rest)
                    (output-stream ?os ?output))

)
```

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

The $ operator makes the variable bind to a *list* of values

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

Different occurrences of the same variable in the LHS: they all bind to the same value

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

A logical *clause*, part of the LHS

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

If the pattern matches, each clause binds to a *fact*, not necessarily distinct

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

Another logical clause. There are two clauses in this LHS.

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

Different occurrences of the same variable define a constraint across clauses, since they all bind to the same value

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

Another constraint

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

This variable binds to the fact that matches the clause

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

The binding of variables propagates from the LHS to the RHS

defrule

(file finitestatemachine.jess)

- It defines a *rule*

Example:

```
(defrule state-transition
    ?current <- (fsm (current-state ?cs)
                     (input-stream ?is $?rest)
                     (output-stream $?output))
    (event (current-state ?cs)
           (input-symbol ?is)
           (output-symbol ?os)
           (new-state ?ns))
     =>
    (printout t "From state " ?cs " input " ?is
                " to state " ?ns " output " ?os crlf)
    (modify ?current (current-state ?ns)
                     (input-stream ?rest)
                     (output-stream ?os ?output))
)
```

More examples

deffacts

(file paritychecker-fsa.jess)

- It is used to define the initial facts, that are in memory at the beginning

  The **(reset)** command erases the working memory and re-asserts all facts defined with **deffacts**

  Example:

```
(deffacts test-string
    (fsm (current-state even)
         (input-stream 0 1 1 1 0 0 1 1)
         (output-stream))
)
```

deffacts

(file paritychecker-fsa.jess)

- It is used to define the initial facts, that are in memory at the beginning

  The **(reset)** command erases the working memory and re-asserts all facts defined with **deffacts**

  Example:

```
(deffacts test-string
     (fsm (current-state even)
          (input-stream 0 1 1 1 0 0 1 1)
          (output-stream))
)
```

In this particular case, just an individual fact is asserted

deffacts

(file paritychecker-fsa.jess)

- It is used to define the initial facts, that are in memory at the beginning

  The **(reset)** command erases the working memory and re-asserts all facts defined with **deffacts**

  Example:

```
(deffacts test-string
     (fsm (current-state even)
          (input-stream 0 1 1 1 0 0 1 1)
          (output-stream))
)
```

In this particular case, just an individual fact is asserted

In general, any number of facts can be asserted via a **deffacts**

deffacts

(file paritychecker-fsa.jess)

- It is used to define the initial facts, that are in memory at the beginning

  The **(reset)** command erases the working memory and re-asserts all facts defined with **deffacts**

  Example:

```
(deffacts test-string
     (fsm (current-state even)
          (input-stream 0 1 1 1 0 0 1 1)
          (output-stream))
)
```

In this particular case, just an individual fact is asserted

In general, any number of facts can be asserted via a **deffacts**

There can be many **deffacts** in the same program

deffacts

(file paritychecker-fsa.jess)

- It is used to define the initial facts, that are in memory at the beginning

  The **(reset)** command erases the working memory and re-asserts all facts defined with **deffacts**

  Example:

```
(deffacts test-string
     (fsm (current-state even)
          (input-stream 0 1 1 1 0 0 1 1)
          (output-stream))
)
```

In this particular case, just an individual fact is asserted

In general, any number of facts can be asserted via a **deffacts**

There can be many **deffacts** in the same program

All of them will be executed by the command **(reset)**

How to load a program

Start Jess first (it improves the effect)

```
$ java jess.Main
Jess>
```

- Loading a file

  (batch paritychecker-fsa.jess)

  also

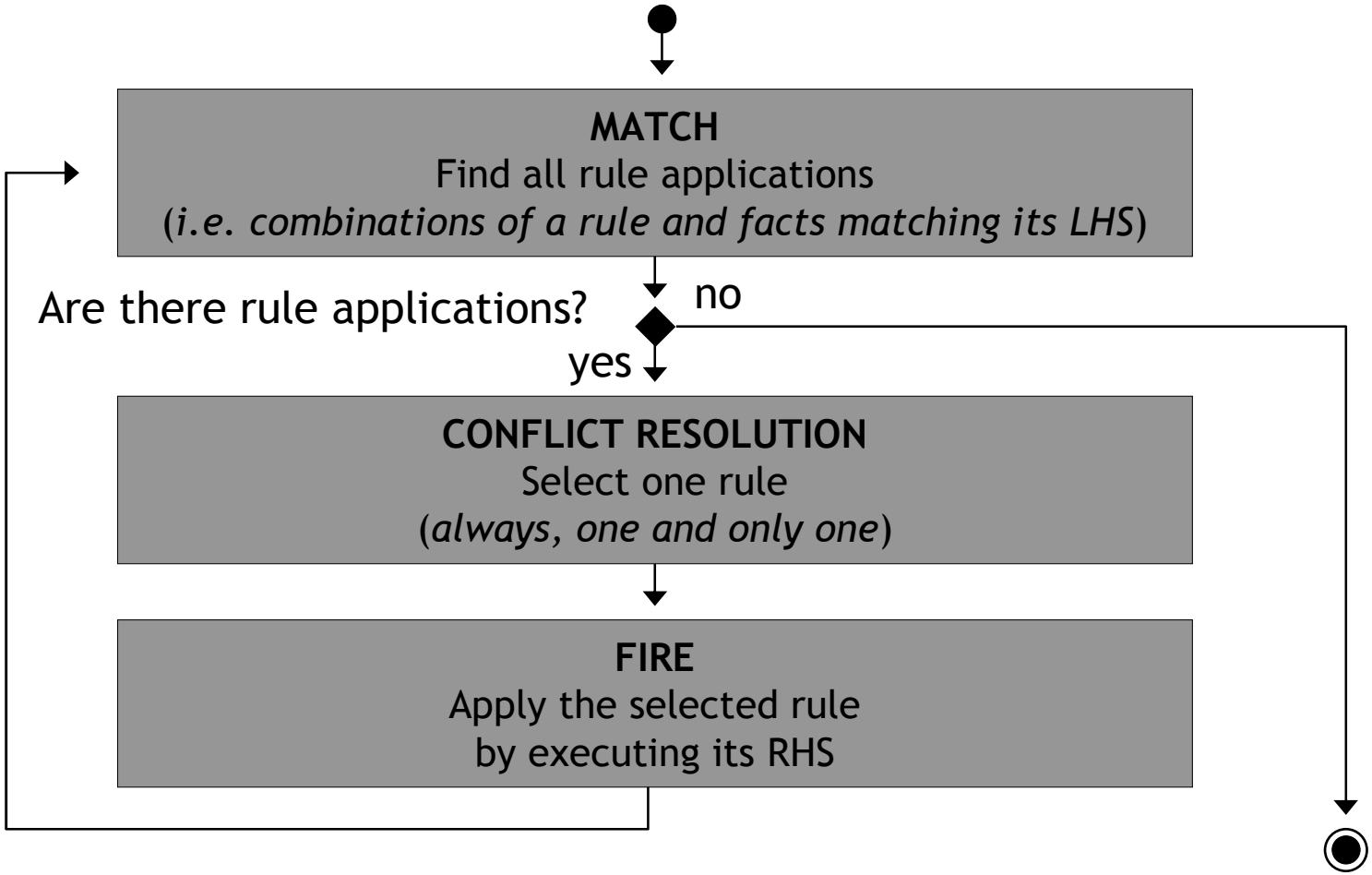
  (batch "paritychecker-fsa.jess")

  If it worked, Jess will say:

  TRUE

How Jess works (in a first approximation)

- Once activated, Jess repeats the same execution cycle



How to start Jess execution

- Reset: erases memory and re-asserts the initial facts

  It causes the execution of **`deffacts`**

  Make sure you always do a “reset” before a new run!

```
(reset)
```

Jess answers:

`TRUE`

- Run: starts the main execution cycle

```
(run)
```

Jess answers:
<here comes the program’s output>

`TRUE`

A few debugging commands

- Execute one cycle at time (i.e. *step mode*)

  `(run 1)`

  Also **`(run`** *n*`)` to execute *n* cycles

- List all facts currently in memory

  `(facts)`

  The answer is not very readable, it takes some effort to decode it

- List all rule applications (*one rule + facts matching its LHS*)

  `(agenda)`

  The answer is not very readable, it takes some effort to decode it