Rules and constructs in CLIPS

Prof. Khaled Shaalan

Khaled.shaalan@buid.ac.ae

Rules

- In an Expert system, the rules are of the form:

IF *certain conditions exists*

THEN *do some actions*

The Components of a Rule

- Rules enable known information to be used to *infer* other information.
- To accomplish work, an expert system must have rules as well as facts.
- Rules can be typed into CLIPS (or loaded from a file).
- Consider the pseudocode for a possible rule:

IF the emergency is a fire

THEN the response is to activate the sprinkler system

Rule Generation - Types

- Relationship
  - **IF** the battery is dead
    THEN the car will not start
- Recommendation
  - **IF** the car will not start
    THEN take a cab
- Directive
  - **IF** the car will not start **AND** the fuel system is ok
    THEN check out the electrical system
- Heuristic
  - **IF** the car will not start **AND** the car is a 1957 Ford
    THEN check the float

Rule Generation – Types

- Vagueness / uncertainty:
  - IF inflation is HIGH
    THEN interest rates might be high
- Re-write using a confidence quantifier :
  - IF inflation is HIGH
    THEN interest rates are high (CQ = 0.8)

Rules

Consist of:

- left-hand side (LHS) or conditions
- Right-hand side (RHS) or actions

Comments are specified with a semicolon

- (test x 5 r 9) ; this is a comment

Instantiated rule:

- LHS of rules is matched by facts in the fact base

Conflict set:

- all instantiations
- multiple instantiations can exist for one rule!

Rules

Conflict resolution strategy picks the rule to execute

Rule firing:

- execution of the RHS of a rule

Refraction:

- part of conflict resolution
- insures that a rules fires only once for the same set of facts

Rule Format

```
(defrule  <rule-name>
      "optional documentation string or comment"
      (<condition-1>)
       (<condition-2>)
              . . .
       (<condition-n>)
  =>
       (<action-1>)
       (<action-2>)
             . . .
       (<action-m>)  )
```

LHS/RHS = Conditions/Actions

- conditions look like facts but:
  - fact’s fields must all be literal
  - condition’s fields can be:
    - Literal
    - wild cards
    - variables
- actions include:
  - assert create new facts
  - retract delete existing facts
  - modify modify existing facts
  - printout display information

An Example Rule

; If <u>animal is </u>a <u>duck </u>

; Then the <u>sound</u> that it makes <u>is quack</u>

```
(defrule duck
"Here comes the quack"
(animalis duck)
=>
(assert (soundis quack)))
```

Analysis of the Rule

- The header of the rule consists of three parts:

1. Keyword *defrule*
2. Name of the rule – *duck*
3. Optional comment string – “Here comes the quack”

- The arrow => represents the beginning of the THEN (RHS) part of the IF-THEN rule.

- The last part of the rule is the list of actions that will execute when the rule fires.

Activation

- If all the patterns of a rule (LHS) match facts (conditions), the rule is activated and put on the agenda.
  - The agenda is a collection of activated rules.

The Agenda and Execution

- To run the CLIPS program, use the *run* command:

  CLIPS> (run [<limit>])↵

  – the optional argument <limit> is the maximum number of rules to be fired – if omitted, rules will fire until the agenda is empty.

Execution/Firing

- When the program runs, the rule with the highest *salience* on the agenda is fired.
- Rules become activated whenever all the patterns of the rule are matched by facts.
- The *reset* command is the key method for starting or restarting .
- Facts asserted by a *reset* satisfy the patterns of one or more rules and place activation of these rules on the agenda.

What is on the Agenda?

- To display the rules on the agenda, use the agenda command:

  CLIPS> (agenda) ↵

- *Refraction* is the property that rules will not fire more than once for a specific set of facts.
- The *refresh* command can be used to make a rule fire again by placing all activations that have already fired for a rule back on the agenda.

Agenda

Suppose a file "duckfile" contains:

(defrule duck (animalis duck) => (assert (soundis quack)))

CLIPS> (assert (animal-is duck))

<Fact-1>

CLIPS> (load "duckfile")

CLIPS> (agenda)

0 duck: f-1

For a total of 1 activation.

CLIPS> (facts)

f-0 (initial-fact)

f-1 (animalis duck)

For a total of 2 facts.

CLIPS> (watch facts)

CLIPS> (run)

==> f1 (soundis quack)

CLIPS> (facts)

f-0 (initial-fact)

f-1 (animalis duck)

f-2 (soundis quack)

For a total of 3 facts.

CLIPS> (agenda)

CLIPS> (run)

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 is asserted
- a pattern entity that did exist before, but was retracted and reasserted (i.e. modified)
  - Take care sometimes lead to infinite loop!

Fact Addresses, Single-Field Wildcards, and Multifield Variables

- A variable can be bound to a fact address of a fact matching a particular pattern on the LHS of a rule by using the pattern binding operator "<-".
- Single-field wildcards can be used in place of variables when the field to be matched against can be anything and its value is not needed later in the LHS or RHS of the rule.
- Multifield variables and wildcards allow matching against more than one field in a pattern.

Variables, Operators, Functions

- Variables name begins with a question mark "?"
- variable bindings
  - variables in a rule pattern (LHS) are bound to the corresponding values in the fact, and then can be used on the RHS
  - all occurrences of a variable in a rule have the same value
  - the left-most occurrence in the LHS determines (generates) the value
  - bindings are valid only within one rule (local scope)
- access to facts
  - variables can be used to make access to facts more convenient:

  ?age <- (age harry 17)

Example: cfile

Suppose a file "cfile" contains:

```
(deffacts initial-colors
    (colors red blue))
(defrule  rule-1
    (colors red brown blue)
    =>  (assert (rule 1 fires)) )
(defrule rule-2
    (colors red blue)
    =>   …)
(defrule rule-3
    (colors blue red)
    =>   …)
(defrule rule-4
    (colors red black)
    =>   …)
(defrule rule-5
    (colors brown red)
    =>   …)
```

And we issue the commands:

```
clips>  (load "cfile")
clips>  (reset)
clips>  (facts)
```

What is in the fact base?

```
f-0  (initial-fact)
f-1  (colors red blue)
```

Suppose that we now:

```
clips>  (run)
```

```
f-0  (initial-fact)
f-1  (colors red blue)
f-2  (rule  2  fires)
```

Example: m-o

```
(defrule assign-orders-to-machine
    ?mach <- (machines id m-1 status idle current-order not-assigned)
    ?orders <- (orders id o-2 status waiting reqd-machine m-1)
    =>
    (retract ?mach ?orders)
    (assert (machines id m-1 status busy current-order o-2))
    (assert (orders id o-2 status assigned reqd-machine m-1)) )
```

Assume we have the facts:

f-1 (machines id m-1 status [illegible]rent-order not-assigned)
f-2 (orders id o-2 status waiti[illegible]qd-machine m-1)

After execution?

f-3 (machines id m-1 status busy current-order o-2)
f-4 (orders id o-2 status assigned reqd-machine m-1)

Example: m-o

```
(defrule assign-orders-to-machine
    ?mach <- (machines id m-1 status idle current-order not-
    assigned)
    ?orders <- (orders id o-2 status waiting reqd-machine m-1)
    =>
    (retract ?mach ?orders)
    (assert (machines id m-1 status busy current-order o-2))
    (assert (orders id o-2 status assigned reqd-machine m-1)) )
```

:And we issue the commands

clips> (load “m-o”)

clips> (rules)

assign-orders-to-machine

a total of 1 defrule

clips> (run)

clips> (clear)

clips> (rules)

What rules and facts we have?

None !

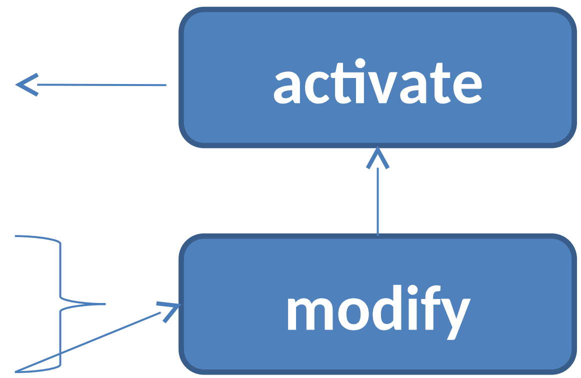
Clear – clears everything to the state when the program was started

Printout

(printout *<log-name>* *<item-1>* … *<item-n>*)

- Performs a write operation
- Output goes to screen if *<log-name>* is t
- crlf forces a carriage return/new line
- Example (printout file):

```
(defrule hello-world-rule
?init <- (initial-fact)
   =>
(retract ?init)
(assert (initial-fact))
(printout t "hello world" crlf) )
```



What happens when this rule executes?

infinite execution!

Top Level Rule Commands

Covered Previously:

(watch <*X*>) -- where <*X*> is rules or activations

(unwatch <*X*>)

New Commands:

(rules) -- displays names of all rules in K.B.

(ppdefrule <*r-name*>) -- display specified rule

(matches <*r-name*>) -- display list of facts that match conditions of rule

(set-break <*r-name*>) -- sets a break point on rule

-- system returns to top level *before* execution of rule

Top Level Rule Commands (cont.)

(remove-break *<r-name>*) -- removes break

(remove-break) -- removes all breaks

(show-breaks) -- display all break points

(agenda) -- display all rule instantiations on agenda

(run) -- start execution, running until no more rules can execute or (halt) is executed

(run *<n>*) -- execute at most *<n>* rules

Salience

- Numeric value associated with rule
- Range for value: -10000 to 10000
- Default value is 0
- Specifies the priority of rule execution
- *Do not (over) use!*

Conflict resolution:

1. **Find rules which are satisfied**
2. **Pick rule with highest salience**
3. **If multiple rules, then apply conflict resolution to pick**

Example

Suppose a file "salience" contains:

```
(deffacts  example-facts
    (a  b  c)
    (Test 1) )


(defrule  r-1
    (a  b  c)
    =>   (assert (rule 1 fires)))

(defrule  r-2
    (declare  (salience  1))
    (a  b  c)
    (Test 1)
    => (assert (rule 2 fires)))

(defrule  r-3
    (declare  (salience  1))
    (a  b  c)
    => (assert (rule 3 fires)))
```

Which rule should be executed first?

either r-2 or r-3, then r-1

Run:

```
CLIPS> (watch facts)
CLIPS> (watch activations)
CLIPS> (reset)
CLIPS> (agenda)
CLIPS> (run)
```

Pattern Matching: Wildcards

?
- single field wildcard
- matches anything in corresponding field of fact

?$
- multi-field wildcard
- matches zero or more fields of a fact

Pattern Matching (cont.)

?<*var*>

- single field variable
- <var> is some word
- this symbol matches anything in the corresponding field of a fact
- value matched is bound to ?<*var*> for scope of rule
- examples: ?cat ?color ?machine

$?<*var*>

- multi-field variable
- matches zero or more fields of a fact
- the value(s) of the matched fields is bound to $?<*var*> for the scope of the rule

Examples

Single field wild cards

LHS Condition	Fact in Fact Base	Match?
(? ?)	(data red)	Yes
(data ?)	(data red)	Yes
(data ?)	(data red green)	NO!
(data ? ?)	(data red green)	Yes
(data red ?)	(data red green)	Yes
(data ? green)	(data green)	NO!

Examples

Multi-field wild cards

LHS Condition	Fact in Fact Base	Match?
($?)	(data red)	Yes
(data $?)	(data red)	Yes
(data red $?)	(data red)	Yes
(data $?)	(data red green)	Yes
(data red green $?)	(data red green)	Yes
($? green)	(data red green)	Yes
($? red $?)	(data red green)	Yes
(data red $?)	(data green red)	NO!
(data $? red $?)	(data green red)	Yes
($? $?)	(data red)	Yes

Examples

Single Field Variables

LHS Condition	Fact in Fact Base	Match?
(data red ?x)	(data red green)	?x=green
(data red ?x)	(data red "green")	?x="green"
(data red ?x)	(data red 17.4)	?x=17.4
(data ?x ?x)	(data red red)	?x=red
(data ?x ?x)	(data red green)	NO!
(data ?x ?y)	(data red green)	?x=red ?y=green
(data ?x ?y)	(data red red)	?x=red ?y=red

Examples

Multi-Field Variables

LHS Condition	Fact in Fact Base	Match?
(data red $?x)	(data red)	$?x=()
(data red $?x)	(data red green)	$?x=(green)
(data red $?x)	(data red one two)	$?x=(one two)
(data $?x $?x)	(data red red)	$?x=(red)
(data $?x $?y)	(data red green)	Multiple matches:
	$?x=()	$?y=(red green)
	$?x=(red)	$?y=(green)
	$?x=(red green)	$?y=()

Example of Rules with Multiple Conditions (file: data1)

Fact Base:

```
(data red green)
(data purple green)
```

Rule Base:

```
(defrule rule-1
    (data red ?x)
    (data purple ?x)
    => (assert (rule 1 fires)) )
(defrule rule-2
    (data red $?x)
    (data purple $?x)
    => (assert (rule 2 fires)) )
```

What rules will be instantiated?

Both rule-1 and rule-2!

Example of Rules with Multiple Conditions (file: data2)

Fact Base:

(data red green) (data red blue green)

(data purple blue) (data purple blue brown)

Rule Base:

```
(defrule rule-3
    (data red ?x)
    (data purple ?x)
    => (assert (rule 3 fires)) )
(defrule rule-4
    (data red $?x)
    (data purple $?x)
    => (assert (rule 4 fires)) )
```

What rules will be instantiated?

None!

Example of Rules with Multiple Conditions (file: data3)

Fact Base:

(data red green) (data red blue green)

(data purple blue) (data purple blue brown)

Rule Base:

```
(defrule rule-5
    (data red ?x)
    (data purple ?y)
    => (assert (rule 5 fires)) )
(defrule rule-6
    (data red $?x)
    (data purple $?y)
    => (assert (rule 6 fires)) )
```

What rules will be instantiated?

One instantiation for rule-5, but 4 for rule-6!

Field Constraints

- In addition to pattern matching capabilities and variable bindings, CLIPS has more powerful pattern matching operators.
- Consider writing a rule for all people who do *not* have brown hair:
  - We could write a rule for every type of hair color that is not brown.
  - This involves testing the condition in a roundabout manner – tedious, but effective.

Field Constraints

- The technique for writing a rule for all non-brown hair colors implies that we have the ability to supply all hair colors – virtually impossible.

- An alternative is to use a field constraint to restrict the values a field may have on the LHS – (the THEN part of the rule.)

Connective Constraints

- Connective constraints are used to connect variables and other constraints.
- *Not connective* – the ~ acts on the one constraint or variable that immediately follows it.
- *Or constraint* – the symbol | is used to allow one or more possible values to match a field or a pattern.
- *And constraint* – the symbol & is useful with binding instances of variables and on conjunction with the *not* constraint.

Field Constraints

Constraints can be placed on the fields of a condition using logical operators:

Syntax: & and | or ~ not

<val1> & *<val2>*	*both values*
<val1> | *<val2>*	*either value*
~*<val>*	*not*
?*<var>* & (*<val1>* | *<val2>*)	*combination*
?*<var>* & ~*<val>*	

Examples

LHS Condition	Fact in Fact Base	Match?
(data ~red)	(data red)	NO
(data ~red&~blue)	(data green)	Yes
(data red|blue)	(data red)	Yes
(data ?col&~red)	(data green)	?col=green
data ?col&~red&~blue)	(data red)	NO
(data ?col&red|blue)	(data red)	?col=red

Combining Field Constraints: person's eye and hair

- Field constraints can be used together with variables and other literals to provide powerful pattern matching capabilities.

- Example #1: ?eyes1&blue|green
  - This constraint binds the person's eye color to the variable, ?eyes1 if the eye color of the fact being matched is either blue or green.

- Example #2: ?hair1&~black
  - This constraint binds the variable ?hair1 if the hair color of the fact being matched is not black.

Sample Rule

```
(defrule  complex-eye-hair-match
              (person  ?name1
                           ?eyes1&blue|green
                           ?hair1&~black)
              (person  ?name2&~?name1
                           ?eyes2&~?eyes1
                           ?hair2&red|?hair1)
   =>
              (printout  t  ?name1  “ has ”  ?eyes1  “ eyes  and ”
                               ?hair1  “ hair”  crlf)
              (printout  t  ?name2  “ has ”  ?eyes2  “ eyes and ”
                               ?hair2  “ hair”  crlf)  )
```

Built-in Functions

Arithmetic:

+ - * / **

Relational:

!= = < <= > >= eq neq

Predicate:

numberp
stringp
wordp
evenp
oddp
integerp

no built-in precedence, only left-to-right and parentheses

Built-in Functions (cont.)

Extended Arithmetic:

Trig. functions, min, max, abs, etc.

I/O Discussed later

Multi-field variable Discussed later

String Discussed later

Notes:

Use &: |: ~: to combine logical functions

Use &= |= ~= to combine computational functions

All functions use prefix notation

Examples

Rule	Facts	Instantiated?
(defrule r-1		
(data ?Y)	(data 3)	?X = 5
(data ?X &:(> ?X ?Y))	(data 5)	?Y = 3
=> . . .)		
(defrule r-2		
(data1 ?Y)	(data2 4)	?Y = 4
(data2 ?X &:(= ?X ?Y))	(data1 4)	?X = 4
=> . . .)		

Examples (cont.)

Rule	Facts	Instantiated?
(defrule r-3 (data1 ?Y) (data2 ?X &:(= ?X ?Y)) => . . .)	(data1 red) (data2 4)	ERROR!
(defrule r-4 (data1 ?Y) (data2 ?X &:(eq ?X ?Y)) => . . .)	(data1 red) (data2 4)	NO!
(defrule r-5 (data1 ?Y) (data2 ?X &:(neq ?X ?Y)) => . . .)	(data2 4) (data1 red)	?Y = red ?X = 4