CLIPS-02

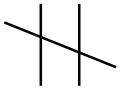
Pattern Matching of CLIPS

By Gwo-Jen Hwang

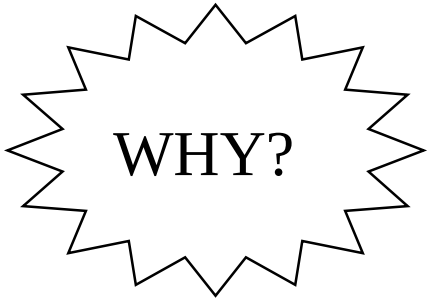
Understanding the concepts of
Expert Systems

+

Knowing how to elicit Expertise



Knowing how to build an
Expert System



Lack of practical concepts and
programming skills.

- variable

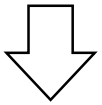
  ?speed

  ?value

No blank between ? and value

IF x is a grandfather

THEN x is a man



```
(defrule grandfather
    (is-a-grandfather ?x)
    =>
    (assert (is-a-man ?x))
)
```

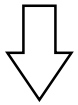
```
(defrule grandfather
        (is-a-grandfather  ?name)
        =>
        (assert (is-a-father  ?name)
                (is-a-man    ?name)
        )
        (printout t  ?name  "is a grandfather" crlf)
)
```


terminal

next line

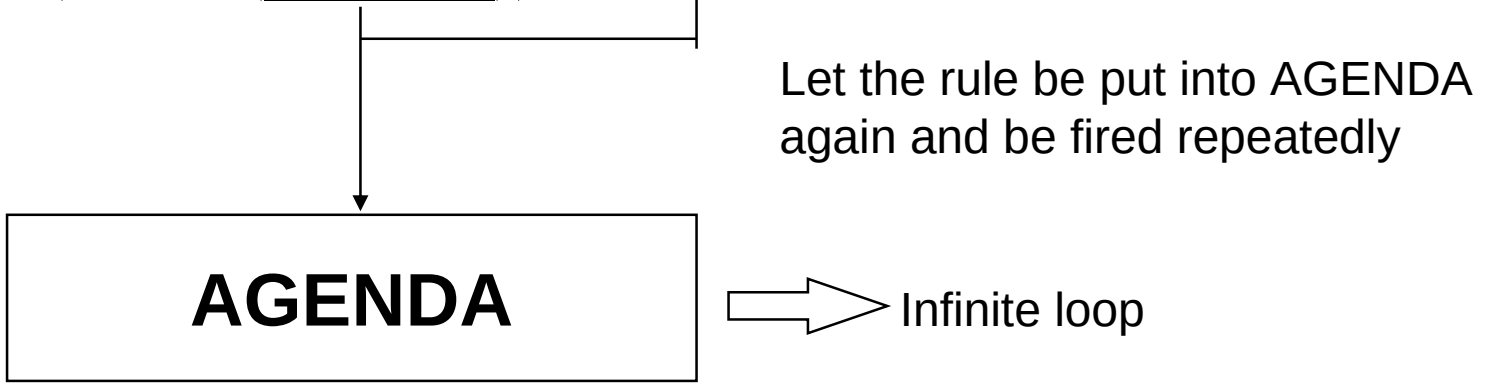
- Facts address and Retraction

```
(defrule modify-grandfather-fact
    (is-a-grandfather  ?name)
   =>                                  f-?  ⇨  fact identifier is needed
    (retract   ?    )                          for retraction
    (assert (has-a-grandson  ?name)
            (is-a-man  ?name)
    )
 )
```



```
(defrule modify-grandfather-fact
     ?old   < - ( is-a-grandfather  ?name)
  =>                 keep fact identifier
     (retract  ?old)
     (assert  (is-a-father  ?name)
              (is-a-man    ?name)
     )
)
```

```
(defrule simple-loop
    ?old-fact  <-  (loop-fact)
    =>
    (printout  t  “Looping?” crlf)
    (retract  ?old-fact)
    (assert  (loop-fact))
)
```



Let the rule be put into AGENDA again and be fired repeatedly

Infinite loop

- Multiple use of variables

```
(defrule  test
        (?x   ?x)
     =>
        (printout  t  “ok” crlf)
 )
```

(John Mary) ⟶ unmatched !

(John John) ⟶ matched !

- Wild cards

```
(defrule  find-brown-haired-people
      (person  ?name   ?    brown)
      =>
      (printout  t  ?name  “has  brown  hair” crlf)
)
(person  John  black  brown)
```

Don’t care

color of hair

?

⇩

Don’t care his eyes’ color

Basic Concepts of applying Expert System Shells:

1. Fact format is user-defined, which should be as easy to read as possible.

John is a employee of IBM.
He is 24 years old.

(employee-of-IBM John 24 male)
or
(John 24 male IBM)

2. Rule format should match fact format.

```
(defrule find-employee-of-IBM
     (employee-of-IBM ?name ?age ?sex)
   =>
     (printout t ?name ?age ?sex)   )
```

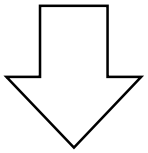
- Multifield wildcards and variables

```
(defrule find-list
      (list  $? )
      =>                    multifield wildcard
      (printout t “found list” crlf)
)
```

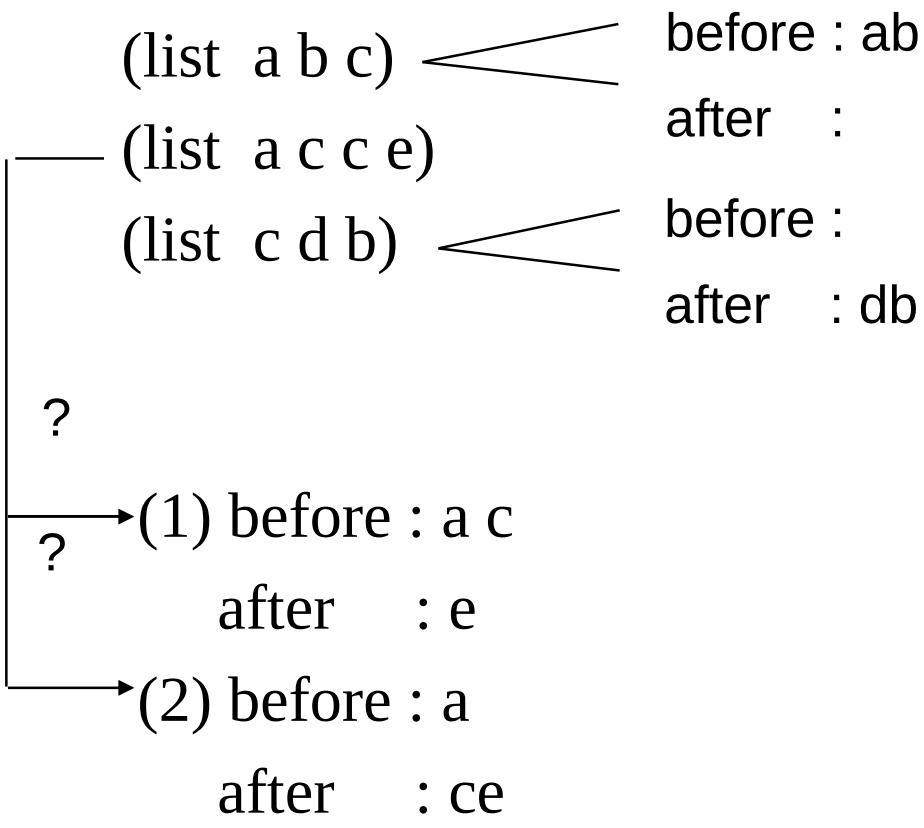
```
(list  a )          matched !
(list  b)           matched !
(list  a b)         matched !
(list  a b c d)     matched !
```

Multifield variable

```
(defrule find-list
      (list  $?item)
     =>
      (printout t “Found a list:” $?item crlf)
)
```



```
(defrule find-item-C
      (list  $?before  C  $?after)
     =>
      (printout t “items before C:” $?before crlf)
      (printout t “item after C:” $?after crlf)
)
```


(list a b c)
before : ab
after :
(list a c c e)
(list c d b)
before :
after : db
?
?
(1) before : a c
after : e
(2) before : a
after : ce

```
(defrule find-item-C
     (list  $?before  C  $?after)
    =>
     (printout t  "before:"  $?before  crlf)
     (printout t  "after:"  $?after  crlf)
)
```

```
(assert (list a c c b))↵
(agenda)↵

        0 : find-item-C : f-1
        0 : find-item-C : f-1

(RUN) ↵
        before : ac
        after  : b
        before : a
        after  : cb
```

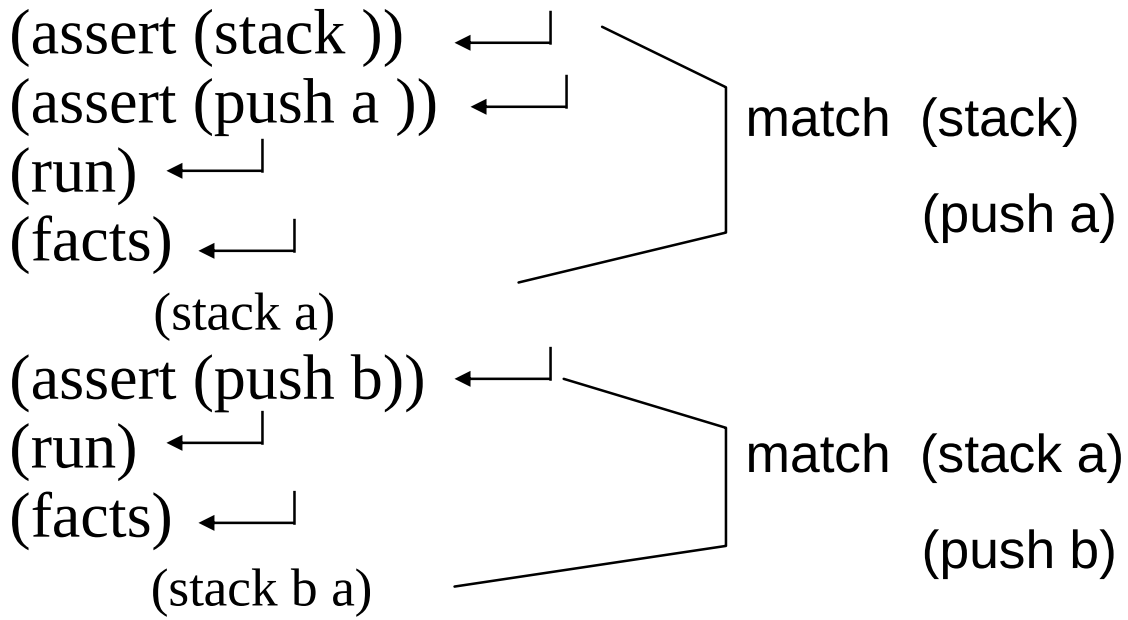
```
(defrule combine-match
      (list  ?  $?  C  ?)
    =>
      (printout t “found!!” crlf)
)
```

```
      (list  c)              unmatched !
      (list  c d)            unmatched !
      (list  a c e)          matched !
      (list  a c b d)        unmatched !
      (list  a d c b)        matched !
```

```
 ?  $?   at least one
    $?   zero or more
     ?   exactly one
```

- Implementing a stack

```
(defrule push-value
      ?push-value < - (push ?value)
      ? stack < -  (stack $?rest)
     =>
      (retract  ?push-value  ?stack)
      (assert (stack ?value $?rest))
)
```



```
(defrule pop-value
        ?push-value < - (pop)
        ? stack < -  (stack ?first $?rest)
       =>
        (retract  ?push-value  ?stack)
        (assert (stack $?rest))
        (printout t " Element " ?first " is removed from the stack" crlf)
        (printout t " Stack: " $?rest crlf)
    )
```

- The **NOT**?field constraint

```
(defrule person-not-brown-hair
    (person  ?name  ?  ~brown)
   =>
    (printout t ?name crlf)
)
```

Not brown

don’t care

eyes hair

(person John black black) matched !

(person Mary black brown) unmatched !

- The ‘**OR**’ field constraint

```
(defrule black-or-brown-hair)
    (person  ?name  ?  brown|black)
   =>
    (printout t ?name crlf)  )
```

OR

The **AND**?field constraint

only brown or black can be matched

```
(defrule black-or-brown-hair
    (person ?name ?   ?color&brown|black)
   =>
    (printout t  ?name "bus" ?color "hair" crlf)
```

bound

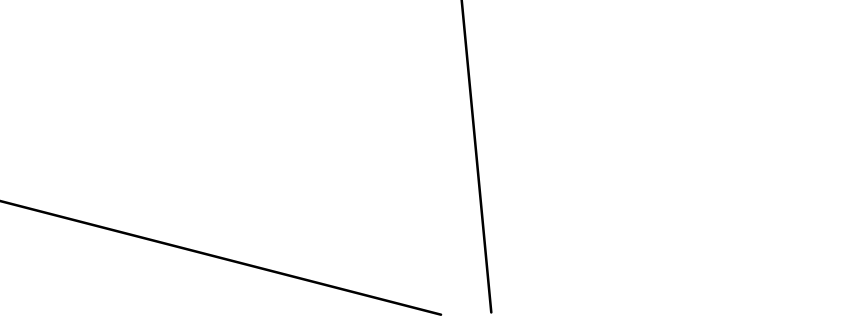
```
(defrule black-and-brown-hair
      (person ?name  ?    black&brown)
     =>
      (printout  t  ? crlf)
)
```

The condition will never be matched.

⇩

a useless rule

```
(defrule black-or-brown hair
    (person ?name ?    ?color&~brown&~black)
    =>
    (printout t  ?name “has”
                 ?color  “hair” crlf)
)
```



only the color which is neither brown nor black can be matched

- Math function

  +

  -

  *

  /

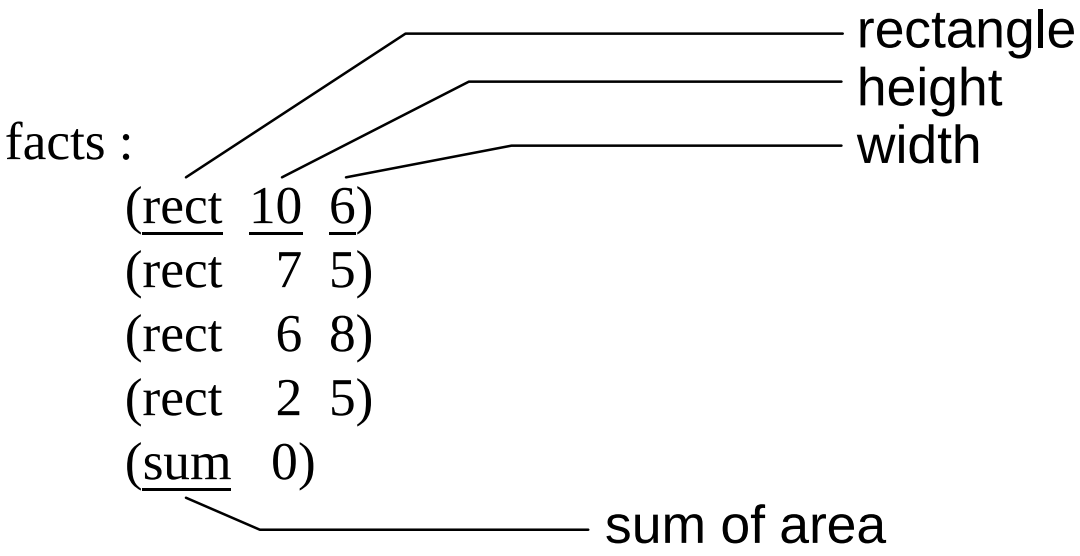
  **

```
CLIPS> (+ 2 2)
       4
CLIPS> (+ 2 (* 3 5))
       17
CLIPS> (assert (answer (+ 2 2)))
        ERROR !!
CLIPS> (assert (answer  =(+ 2 2)))
```

nested lists are not allowed

(answer 4)

‘+’ is treated as an operator




```
(defrule  sum-rect
     (rect  ?h  ?w)
      ?sum   <-   (sum  ?total)
    =>
      (retract   ?sum)
      (assert  (sum =(+ ?total (* ?h ?w))))
)
```

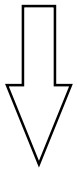
?total = ? total + ?h * ? w

Infinite loop !! WHY?

One way :

```
(defrule  sum-rect
 ?x <-  (rect ?h  ?w)
          ?sum  <-  (sum  ?total)
        =>
          (retract ?x ?sum)
          (assert (sum =(+ ?total (* ?h?w))))
)
```

```
(rect   10  6)
(rect    7  5)
(rect    6  8)
(rect    2  5)
(sum    0   )
```



(Run)

Not a good idea
Since the facts (rect ? ?)
should remain the same

```
(sum   153 )
```

An alternative approach:

```
(defrule  sum-rect
     (rect   ?h  ?w)
    =>
     (assert  (area =(*  ?h  ?w)))
)


(defrule  sum-area
      ?sum  <-  (sum  ?total)
      ?new-area  <-  (area  ?area)
     =>
      (retract  ?sum  ?new-area)
      (assert  (sum  =(+ total ?area)))
)
```