CLIPS
(C Language Integrated Production System)

Rule-based programming language
Based on OPS-5
Not biased towards any particular data representation
Uses a general representation -- no preset interpretations for any symbols

Interpreter uses \textit{recognize-act} cycle:

1. \textit{Match} - find all rules with matched antecedents
   a. each combination of facts that satisfies a rule is called an \textit{instantiation}
   b. each matching rule is added to the \textit{agenda}
2. \textit{Conflict Resolution} - select a rule from the agenda to execute. If none, halt.
3. \textit{Act} - execute rule performing specified actions
4. \textit{Repeat} - go back to step 1.
A Production System Cycle

1. Pattern Matching
   - asserts/retracts/modify facts
   - change rules

2. Conflict Resolution
   - select rule

3. Fire rule

User's program

Knowledge-Base (Rules)

Working Memory (Facts)
CLIPS Data Base

Consists of a list of facts
Each fact:
  • consists of one or more fields enclosed by parentheses
    
    \[(x \ y \ z)\]
    \[(> \ 3 \ 1)\]
    \[(pen \ color \ red)\]
  • represents a piece of information
  • used to represent the current state of the problem
  • is declarative knowledge
CLIPS Data Base (cont.)

Notes:

- First field in a fact should express a relationship
- Initial state of a problem (or the problem domain) is defined by a `deffacts`
- New facts are added by `assert`
- Re-asserted facts are ignored - *Refraction*
- Old facts are removed by `retract`
Fields of a Fact

Words

• A sequence of alphabetic, numeric, underscores, or dash symbols
  
  $\textit{foo}$
  $\textit{foo-bar}$
  $\textit{foo\_bar}$
  $\textit{foo-12}$

• Note that CLIPS is case sensitive
  
  $\textit{foo}$ is not the same as $\textit{FOO}$ or $\textit{Foo}$
Fields of a Fact (cont.)

Strings
  • A collection of characters within double quotation marks
  • Strings and words are not equivalent
    "cat" is not the same as cat

Numbers
  • Are stored in a single precision, floating point representation
    237  237.0  2.37E+2
    are all the same!
Example Facts

(foo 1286 “this is field 3”)
(this is a fact with 7 fields)
(“This is a facts with 1 field - a string”)
(animal-is walrus)
(animal-is duck)
(animals-are duck horse cow)
(said duck “quack”)
(address 1000 main st)
(address 1000 “main st”)
(address “1000 main st”)
Facts

Use the first field to describe relationship between subsequent fields:

\[(<relation> <field-1> <field-2> \ldots)\]

Use object-attribute-value and attribute-value formats:

\[(\text{person l-name smith} \n \text{f-name john} \n \text{ssn 123457689} \n \text{dept engineering})\]
Asserting and Retracting Facts

Actions:

clips> (reset)
clips> (assert (plays ivan tennis))
clips> (assert (plays martina tennis))
clips>

The data base:

f-0 (initial-fact)

f-1 (plays ivan tennis)

f-2 (plays martina tennis)
Asserting and Retracting Facts

Actions:

clips> (reset)
clips> (assert (plays ivan tennis))
clips> (assert (plays martina tennis))
clips> (retract 1)
clips> (assert (plays martina tennis))
clips>

The data base:

f-0 (initial-fact)
f-1
f-2 (plays martina tennis) No Change!
Linking Facts through Common Fields

Sometimes it is advantageous to create multiple facts logically linked together by a common field:

(person
  ssn 123456789
  l-name smith
  f-name john
  dept engineering)

(personal
  ssn 123456789
  age 31
  height 71
  weight 175
  sex male
  m-status single)

(financial
  ssn 123456789
  salary 45000
  title senior-engineer)
Why Link Facts through Common Fields?

(machine id m-1 status idle cur-order none)
(machine id m-2 status idle cur-order none)
...
(machine id m-10 status idle cur-order none)
(order id o-1 status waiting requires m-1)
...
(order id o-9 status waiting requires m-8)

Now envision rules:

IF order waits for a specific machine and the machine is idle
THEN assign order to machine and change status of order to assigned and change status of machine to busy
Control of Fact Base

Deffacts
used to define a group of facts acting as initial data (knowledge) for a problem

(deffacts <deffacts-name>
  “optional comment”
  ( <fact-1> )
  ( <fact-2> ) ...
  ( <fact-n> ) )

Load
used to load file containing knowledge base into memory. It does not execute the deffacts statement(s)

(load “myfile-name”)

I-13
Control of Fact Base (cont.)

Reset

takes all facts contained in the deffacts statements and enters them into the fact base
(reset)

Undeffacts

removes specified deffacts statement from memory

(undeffacts <deffacts-name>)

Note that CLIPS provides a special deffacts statement:

(deffacts initial-fact
  (initial-fact) )
How Do These Work

Load places information from a file into memory storage
Reset takes information in memory storage and creates material needed for program
Undeffacts removes information from memory storage

(load "file")
How Do These Work (cont.)

Load places information from a file into memory storage
Reset takes information in memory storage and creates material needed for program
Undeффacts removes information from memory storage

(load “file”)
(reset)
How Do These Work (cont.)

Load places information from a file into memory storage
Reset takes information in memory storage and creates material needed for program
Undefferacts removes information from memory storage

(load "file")
(reset)
(undefferacts "X")
How Do These Work (cont.)

Load places information from a file into memory storage
Reset takes out information in memory storage
Undeффacts removes information from memory storage

(load "file")
(reset)
(undeффacts “X”)
(reset)
Example

Assume the following is in the file “mfile”:

(deffacts initial-machine-configs
  (machine id m-1 status idle)
  (machine id m-2 status idle)
  (machine id m-3 status idle)

We now enter the following commands:

clips> (load “mfile”)
clips> (reset)
clips> (facts)

What’s in memory?

f-0 (initial-fact)
  f-1 (machine id m-1 status idle)
  f-2 (machine id m-2 status idle)
  f-3 (machine id m-3 status idle)
Example (cont.)

Now we enter:

clips> (undeffacts initial-machine-configs)
clips> (facts)

What’s in memory? f-0 (initial-fact)
               f-1 (machine id m-1 status idle)
               f-2 (machine id m-2 status idle)
               f-3 (machine id m-3 status idle)

What about now entering:

clips> (reset)
clips> (facts)

What’s in memory?  f-0 (initial-fact)
                   f-1
                   f-2
                   f-3
Other Useful Commands

(watch  <item>)
  causes display of changes in  <item>  which can
  be activations, facts, rules, all, etc.

(unwatch  <item>)
  turns off display of specified  <item>

(dribble  <file>)
  sends all displayed output also to specified file

(dribble-off)
  stops output to file

(close  <file>)
  closes specified file
Logging a Session

c: clips
clips> (dribble-on "lfile")
    ...
    *various commands of the session*
    ...
clips> (dribble-off)
clips> (close "lfile")
clips> (exit)
c:


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!
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 rule fires only once for the same set of facts
Rule Format

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

• LHS conditions are matched to fact base to determine if rule is eligible to fire
• being eligible does not guarantee a rule will fire
• conditions look like facts but:
  fact’s fields must all be literal
  condition’s fields can be:
  - literal
  - wild cards
  - variables
RHS Actions

• Typical RHS actions include:
  
  - assert create new facts
  - retract delete existing facts
  - printout display information

• Not practical to retract facts by referring to their index number
  
  instead, refer to them using a conditional variable -- word prefaced by a ?

  (defrule rule-1
    ?init <- (initial-fact)
    =>
    (retract ?init) )