Chapter Overview

Introduction  CLIPS overview

Notation
    similar to regular expressions

Facts
    elementary statements

Rules
    relations between statements

Variables, Operators, Functions
    advanced pattern matching

Input/Output
    getting knowledge into and out of CLIPS

Program Execution

User Interface
    command line or GUI
Introduction

CLIPS stands for

$C$ Language Implementation Production System

forward-chaining

starting from the facts, a solution is developed

pattern-matching

Rete matching algorithm: find “fitting” rules and facts

knowledge-based system shell

empty tool, to be filled with knowledge

multiparadigm programming language

rule-based, object-oriented (COOL) and procedural
close to LISP

symbols, characters, keywords
	entered exactly as shown: (example)

square brackets  [...]  
	contents are optional: (example [test])

less than / greater than  < ... >  
	replace contents by an instance of that type  
	(example  <char>)

star  *  
	replace with zero or more instances of the  

type  <char>*

plus  +  
	replace with one or more instances of the type  
	<char>+ (is equivalent to  <char>  <char>* )

vertical bar  |  

type among a set of items:  true  |  false
Tokens and Fields

tokens

groups of characters with special meaning for CLIPS, e.g. ( ) \ separated by delimiters (space, tab, Carriage Return, ...)

fields

particularly important group of tokens

CLIPS primitive data types

- float
decimal point 1.5 or exponential notation
  3.7e10

- integer
  [sign] <digit>+ 

- symbol
  <printable ASCII character>+ 
e.g. this-is-a-symbol, wrz1brmft, !?@*+
• string
  delimited by double quotes
e.g. "This is a string"

• external address
  address of external data structure
  returned by user-defined functions

• instance name (used with COOL)
  delimited by square brackets

• instance address (used with COOL)
  return values from functions
entering CLIPS

double-click on icon, or type program name system prompt appears:
CLIPS>

exiting CLIPS

at the system prompt
CLIPS>
type (exit)
Note: enclosing parentheses are important;
they indicate a command to be executed, not just a symbol
Facts

elementary information item

relation name
  symbolic field used to access the information

slots  (zero or more)
  symbolic fields with associated values

deftemplate construct
  used to define the structure of a fact (names and number of slots)

deffacts
  used to define initial groups of facts
Examples

of facts

ordered fact

(person-name Franz J. Kurfess)

deftemplate fact

(deftemplate person "deftemplate example"
  (slot name)
  (slot age)
  (slot eye-color)
  (slot haircolor))
an instance of a fact is created by

(assert (person (name "Franz J. Kurfess")
  (age 40)
  (eye-color brown)
  (haircolor brown)))

initial facts

(deffacts kurfesses "some members of the Kurfess family"
  (person (name "Franz J. Kurfess") (age 40)
    (eye-color brown) (haircolor brown))
  (person (name "Hubert Kurfess") (age 39)
    (eye-color blue) (haircolor blond))
  (person (name "Bernhard Kurfess") (age 36)
    (eye-color blue) (haircolor blond))
  (person (name "Heinrich Kurfess") (age 33)
    (eye-color brown) (haircolor blond))
  (person (name "Irmgard Kurfess") (age 32)
    (eye-color green) (haircolor blond)))
of facts

adding facts
  (assert <fact> +)

deleting facts
  (retract <fact-index> +)

modifying facts
  (modify <fact-index> (<slot-name> <slot-value>)+ )
retracts the original fact and asserts a new, modified fact

duplicating facts
  (duplicate <fact-index> (<slot-name> <slot-value>)+ )
adds a new, possibly modified fact
inspection of facts

(facts)

prints the list of facts

(watch facts)

automatically displays changes to the fact list
components of rules

general format

( defrule <rule name> ["comment"]
   <patterns>* ; left-hand side (LHS)
   ; or antecedent of the rule
   =>
   <actions>* ) ; right-hand side (RHS)
   ; or consequent of the rule

rule header

   defrule keyword, name of the rule, optional
   comment string

rule antecedent (LHS)

   patterns to be matched against facts

rule arrow

   separates antecedent and consequent

rule consequent (RHS)

   actions to be performed when the rule fires
Examples

of rules

simple rule

(defrule birthday-FJK
   (person (name "Franz J. Kurfess")
      (age 40)
      (eye-color brown)
      (haircolor brown))
   (date-today April-13-97)
=>
   (printout t "Happy birthday, Franz!")
   (modify 1 (age 41)))

very limited:

• LHS must match facts exactly
• facts must be accessed through their index number
• changes must be stated explicitly
Variables, Operators, Functions

for enhanced pattern matching capabilities

variables

- symbolic name beginning with a question mark "?"
- 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 must have the same value
- the first (left-most) occurrence in the LHS determines the value
- bindings are valid only within one rule
- variables can be used to make access to facts more convenient:
  ?age <- (age harry 17)
wildcards

the question mark "?" matches any single field within a fact
the multifield wildcard "$?" matches zero or more fields in a fact

field constraints

- **not** constraint " "
  the field can take any value except the one specified
- **or** constraint " | "
  specifies alternative values, one of which must match
- **and** constraint " & "
  the value of the field must match all specified values
 mostly used to place constraints on the binding of a variable
**mathematical operators**  
basic operators (+,-,*,/) and many functions (trigonometric, logarithmic, exponential) are supported  
prefix notation  
no built-in precedence, only left-to-right and parentheses

**test feature**  
evaluates an expression in the LHS instead of matching a pattern against a fact

**pattern connectives**  
multiple patterns in the LHS are implicitly AND-connected  
patterns can also be explicitly connected via and, or, not

**user-defined functions**  
external functions written in C or other languages can be integrated
Examples

of rules

more complex rule

(defrule find-blue-eyes
 (person (name ?name)
   (eye-color blue))
 =>
 (printout t ?name " has blue eyes."
 CRLF))
rule with field constraints

(defrule silly-eye-hair-match
  (person (name ?name1)
   (eye-color ?eyes1|blue|green)
   (hair-color ?hair1|black))
  (person (name ?name2|?name1)
   (eye-color ?eyes2|eyes1)
   (hair-color ?hair2|red|hair1))
=>
  (printout t ?name1 " has " ?eyes1 " eyes
   and " ?hair1 " hair." CRLF)
  (printout t ?name2 " has " ?eyes2 " eyes
   and " ?hair2 " hair." CRLF))
Manipulation of Constructs

show list of constructs
(list-defrules), (list-deftemplates),
(list-deffacts) prints a list of the respective constructs

show text of constructs
(ppdefrule <defrule-name>),
(ppdeftemplate <deftemplate-name>),
(ppdeffacts <deffacts-name>) displays the text of the construct (“pretty print”)

deleting constructs
(undefrule <defrule-name>),
(undeftemplate <deftemplate-name>),
(undeffacts <deffacts-name>) deletes the construct (if it is not in use)

clearing the CLIPS environment
(clear) removes all constructs and adds the initial facts to the CLIPS environment
print information

(printout <logical-device> <print-items>*
logical device frequently is the standard
output device t (terminal)

terminal input

(read [<logical-device>])
(readline [<logical-device>])
read an atom or string from a logical device
the logical device can be a file which must be
open

open / close file

(open <file-name> <file-ID> [<mode>])
(close [<file-ID>])
open /close file with <file-id> as internal
name
load constructs from file
  (load <file-name>) \(^1\)

save constructs to file
  (save <file-name>) saves all current
  constructs to the file

\(^1\)backslash \ is a special character and must be “quoted”
(preceded by a backslash \)
e.g. (load "B:\\clips\\example.clp")
Program Execution

execution of rules

agenda
  if all patterns of a rule match with facts, it is
  put on the agenda
  (agenda) displays all activated rules

salience
  indicates priority of rules

refraction
  rules fire only once for a specific set of facts
  (refresh <rule-name>) reactivates rules
**execution** of a program

- **(reset)** prepares (re)start of a program: all previous facts are deleted
  initial facts are asserted
  rules matching these facts are put on the agenda

- **(run [<limit>])** starts the execution

- breakpoints
  **(set-break [<rule-name>])** stops the execution before the rule fires, continue with **run**
  **(remove-break [<rule-name>])**
  **(show-breaks)**
Watching

facts, rules, activations, ...

watching the execution
(watch <watch-item>) prints messages about activities concerning a <watch-item> (facts, rules, activations, statistics, compilation, focus, all)
(unwatch <watch-item>) turns the messages off

facts
assertions (add) and retractions (delete) of facts

rules
message for each rule that is fired

activations
activated rules: matching antecedents these rules are on the agenda
statistics
   information about the program execution
   (number of rules fired, run time, …)

compilation  default
   constructs loaded by the (load) command

focus
   used with modules
User Interface

interaction with CLIPS

menu-based version

most relevant commands are available through windows and menus
Chapter Review

Introduction  CLIPS overview

Notation
    similar to LISP, regular expressions

Facts
    (deftemplate), (deffacts)
    assert / retract

Rules
    (defrule ...), agenda

Variables, Operators, Functions
    advanced pattern matching

Input/Output
    (printout ...), (read ...), (load ...)

Program Execution
    (reset), (run), breakpoints

User Interface  command line or GUI