CLIPS Introduction

Based on cs367-04 (CLIPS).pdf by Ian Watson

Junli Tao

jtao076@aucklanduni.ac.nz

Rm 321-723, Tamaki Campus

(Wed. 1:00pm-2:00pm)

Facts

- Create: `(assert [facts])`
- `(facts) (retract ) (clear)`
- Facts may be a list of atoms
  - `(foo bar baz)`
  - `(foo) (bar) (baz)`
- Facts may be
  - `(size 3.5)`
  - `(mood happy) (mood grumpy)`
  - `(hand player1 AD QC 3H)`
- Symbols are any sequence of ascii characters
  - May not begin with $? or ?
  - May begin with < but not contain it

Rules

- Use matching to decide if a rule can be fired
- `(defrule [conditions] => [results])`

- If the left hand side of a rule is satisified, that rule is “fired”
- Basic method is satisfying a fact exists

Rule Examples

```
(defrule see
  (name frank)
 =>
  (assert (seen frank))
 )

(defrule greet
  (seen frank)
 =>
  (assert (greet frank))
 )
```

Rule Firing

- A rule that can fire is activated
- A rule is activated when all the left hand side condition are met
- How come rules don’t fire twice if the facts are not altered?
  - All facts are “stamped” with the time of creation
  - All rules are “stamped” with the time it last fired
  - If all the times for the facts on the left hand side are earlier than the time the rule last fired, the rule cannot activate

Execution Handling

- `(run)` starts the inference engine
  - `(run n)` fire the next `n` rules
- `(agenda)` shows the rules that can be executed
- `(rules)` list all the rules in memory

strategies

- Depth: newer activated rules first
- Breadth: newer activated rules last
- Simplicity: rules with less conditions first
- Complexity: more conditions first
- LEX: fire rules that use more recent facts
- MEA: uses the time tag of the first element
- Random: select randomly

salience

- Tells the engine what the priority of a rule is (0-255)

```
(defrule is-silly (declare (salience 10))
    (is-silly ?x) =>
    (printout t ?x “ is a silly thing” crlf) )
(defrule is-still-silly (declare (salience 5))
    (is-silly ?x) =>
    (printout t ?x “ is still silly” crlf) )
```

Variables

- ?* or $?*
  - `?name`
  - `$?stuff-and-things`
- `Used in functions,rules`

Binding

- Use variables to test values and to pass information
- Explicit binding
  - `(bind ?percent-chance (random 1 100))`
- Implicit binding
  - `(fact ?name)`
  - Binds `(fact a),(fact b)`

Functions

- Used to compute simple values
  - Not as useful as rules, but simpler to describe
- (deffunction [name] ([arglist]) [action])

Example

```
(deffunction fibbonacci (?f)
   (if (or (= ?f 1) (= ?f 0) )then
        1
   else
        (+    (fibbonacci (- ?f 1) )
              (fibbonacci (- ?f 2) )
              )
        )
   )
```

Built in Functions

- `(+ )  (- )  (* )  (\ )  (** )  (mod )`
- `(= )  (< )  (> )  (<= )  (>= )  (<> )`
- `(and ) (or ) (not )`
- `(random min max)`
- `(sin ) (cos ) (tan ) (sqrt )`
- `(printout t “hello” crlf)`
- `...`

i/o functions

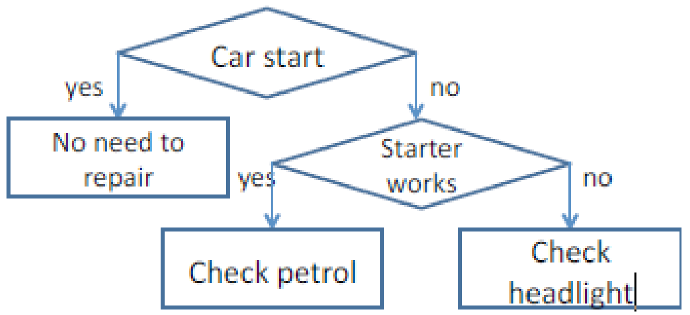
- `(printout t “hello” crlf)`
  - Output to the terminal
- `(read)`
  - Returns a single element
- `(readline)`
  - Returns a string

files

- Save rules
  - `(save “car_test.clp”)`
  - `(load “car_test.clp”)`
- Save facts
  - `(save-fact “car_test.facs”)`
  - `(load-fact “car_test.facs”)`

Example

- Implement the decision tree in CLIPS (“car_test.clp”)



```
(defrule ~~MAIN::~~carDiagnosis
   =>
   (printout t "does car start?(1-yes,0-no)" crlf)
   (bind ?x (read))
   (if (= ?x 1)
      then
      (assert (car-turn-on yes))
      else
      (assert (car-turn-on no))))

(defrule ~~MAIN::~~starterOn
   (car-turn-on yes)
   =>
   (printout t "solution: no need to repair." crlf))

(defrule ~~MAIN::~~starterOff
   (car-turn-on no)
   =>
   (printout t "does starter work?" crlf)
   (bind ?x (read))
   (if (= ?x 1)
      then
      (printout t "solution:check petrol." crlf)
      else
      (printout t "solution: check headlight." crlf)))
```

deffacts

- Can contain anything you can assert
- Only asserted when the engine is reset

Example

```
(deffacts nice “stuff that is tasty”
  (nice watermelon)
  (nice fudgecake)
  (nice burgers))

(deffacts nasty “stuff that is yuck”
  (foul old-cabbage)
  (foul slimy-fungus)
  (foul apricot-and-chicken))
```

Templates

- Used to create objects (nested facts, complex information)
- `(deftemplate [name] [comment]`
  `[list of attributes])`
- Simplifies related facts
- Makes facts more uniform, more structured
- Order of attributes irrelevant

Example

```
(deftemplate person "information about a
  person"
   (slot name)
   (slot gender (allowed-symbols M F N))
   (slot age (type NUMBER))
   (multislot friends)
)

(assert (person (name Daniel) (age 24)
   (gender M) (friends Simon Jane)))
```

Another Example

```
(deftemplate critter "taxonomic info"
  (slot domain)(slot kingdom)
  (slot phylum)(slot class)
  (slot order) (slot family)
  (slot genus) (slot species))
```

Giant Atlantic Squid:

```
(critter (domain eukarya) (kingdom animalia)
  (phylum mollusca) (class cephalopoda)
  (order tuethida) (family Architeuthidae)
  (genus Architeuthis ) (species dux))
```

Car Frame or Template Example

Slots	Fillers
Name	car name
Type	sedan, sports, station_wagon . .
Manufacturer	GM, Ford, Chrysler, Toyota . . .
Owner	Name of owner
Wheels	4
Transmission	manual, automatic
Engine	petrol, diesel, hybrid
Condition	lemon, OK, peach
Under-warranty	no, yes

Car Instance Example

Slots	Fillers
Name	Alice’s car
Type	station_wagon
Manufacturer	GM
Owner	Alice Apple
Wheels	4
Transmission	manual
Engine	petrol
Condition	OK
Under-warranty	yes

Fact References

- `?name <- (fact)`
- `(retract ?name)`
  - Opposite of `(assert)`, removes from factlist
- `(modify ?name (new-fact))`
  - Changes fact to new-fact

Debugging in CLIPS

- Use the fact list and the agenda to work out what is firing (compared to what should be firing)
  - Agenda is sorted according to salience
  - If any rules can be fired and have the same salience: a strategy is used to resolve the conflict.

- Use `(run 1)` to step through rules

- `(watch [all,facts,rules,activations])`
  - Can specify template/rule names to watch
- `(set-break rule-name)`
  - Stops before rule is executed
- `(dribble-on file-name)`
  - Outputs a trace (all facts and rule firings) to the specified file

Examples

- Sources available (http://www.cs.auckland.ac.nz/courses/compsci367s2c/resources/clips/examples/)
  - Auto.clp
  - Animal.clp
  - Usedcar.clp
  - Stove.clp
  - …