


CompSci.367
The Practice of Artificial Intelligence

CLIPS
 Assoc. Prof. Ian Watson


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Overview

- CLIPS is a programming language that provides support for rule-based, object-oriented and procedural programming.
- The search in the Inference Engine uses forward chaining and rule prioritization.
- Looks like LISP (List Processing) with object features
- Developed by NASA in the 80s


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Object Features

- CLIPS Object-Oriented Language (COOL) is a hybrid of features found in Common Lisp Object and SmallTalk.
- Example: object template or frame for a bearing
 - (deftemplate bearing (slot type) (slot size) (slot load) (slot lubrication) (slot max_temperature). . .)


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



New Paradigm

- NOT LIKE JAVA / C++ !
- Provide information on what to do not how to do it.
 - (no algorithms to define)
- (-ve) Can take a little getting used to
- (+ve) Very small code size to do complicated adaptable things


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Facts

- Create: (assert [facts])
- (facts) (clear)
- Facts may be a list of atoms
 - (foo bar baz)
 - (foo (bar baz))
- Facts may be anything (**symbols, not variables**)
 - (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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Example

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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Templates

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

© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz




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)))
```

© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz




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))
```


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



deffacts

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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



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, 6
Transmission	manual, automatic
Engine	gasoline, diesel, methanol
Condition	lemon, OK, peach
Under-warranty	no, yes


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Car Instance Example

Slots	Fillers
Name	Alice's car
Type	station_wagon
Manufacturer	GM
Owner	Alice M. Agogino
Wheels	4
Transmission	manual
Engine	gasoline
Condition	OK
Under-warranty	yes

© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz




example

```
(defacts nice "stuff that is tasty"
  (nice watermelon)
  (nice fudgecake)
  (nice burgers))

(defacts nasty "stuff that is yuck"
  (foul old-cabbage)
  (foul slimy-fungus)
  (foul apricot-and-chicken))
```


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Typing in clips

- Not strictly typed (symbols not variables)
- Atoms (any single datum)
- Performs run-time checking on type
- `(* a 3)` gives an error
- Interpreted therefore no compilation errors


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Playing with the engine

- `(facts)` list the current facts
- `(clear)` clear the engine
 - Clears facts, templates, rules and functions
- `(reset)` clear all the current facts
 - Asserts all deffacts and another fact called `(initial-fact)`


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



recap

- CLIPS
 - Basics
 - Facts
 - Templates
 - Deffacts
 - Execution control


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Variables

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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Functions

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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Example

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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



In built functions

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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Rules

- Use matching to decide if a rule can be fired
- (defrule [conditions] => [results])
- If the left hand side of a rule is satisfied, that rule is "fired"
- Basic method is satisfying a fact exists

© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz




example

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

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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Binding

- Use variables to test values and to pass information
- (fact ?name)
 - Binds (fact a), (fact b)

© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz




Using Bindings

- Explicit binding
 - `(bind ?percent-chance (random 1 100))`
- Bindings in test cases


```
(defrule check-if-positive
  (value ?v)
  (> ?v 0)
  =>
  (assert (is-positive ?v)) )
```
- If a fact exists the binding evaluates to TRUE
- Implicit conjunction on the LHS


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Fact references

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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Recap

- Variables
- Functions
- Rules
- Binding
- Fact References

© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz




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) )
```


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



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 can activate


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



The Agenda

- The agenda contains all rules that can be fired
- 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.


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Debugging in CLIPS

- Use the fact list and the agenda to work out what is firing (compared to what should be firing)
- 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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



i/o functions

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


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



files

- `(open "file.foo" file-handle "r")`
 - `(readline file-handle)`
 - "r" = read, "w" = write, "a" = append, "r+" = read and append, "wb" = write binary
- `(close file-handle)`
 - Closes the file stream


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Fact \Leftrightarrow String functions

- Useful for converting input to facts
 - `(create$ (list of stuff))`
 - Creates the list "on the fly"
 - `(explode$ (create$ this that the other thing))`
 - "this that the other thing"


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



recap

- Methods for processing lists
- Sallience
- How rules are chosen to execute
- Debugging
- I/O and strings


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



JESS

CLIPS + java
(since you've probably noticed
CLIPS has no interface)


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Quick Note

- JESS 6.0 does NOT WORK WITH J2SDK 5.0
- Assert is now a keyword
 - You could alter the source and delete the assert methods since we aren't using them
- 30 day trialware (can download for free from the university)


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Importing and compiling JESS

- Package jess
 - jess.* is all you need
 - Also jess.awt.* and jess.factory.* (not very useful)
- Must have jess.jar in the classpath (JESS 6.1)
 - javac -cp jess.jar;. File.java
 - java -cp jess.jar;. File
- Must deal with JessException

© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz




The Rete class

- Holds the CLIPS script
- No parser of its own (use Jesp class)

```

Rete infEngine = new Rete();
Jesp parser = new Jesp(new FileReader("me.clp"),
    r);
Parser.parse();
// can now use the rete class
  
```


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Execution of the Rete class

- Managing execution from java
 - Rete.run()
 - Rete.run(int max)
 - Rete.reset()
 - Rete.clear()
 - Rete.halt()
- Same as the corresponding CLIPS functions

© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Manipulating the Rete class

- Adding Facts


```


Fact f = new Fact("fact", r);
r.assertFact(f);
r.assertString("(this is a fact)");
      
```

 - Setup values from the java program
- Removing facts


```

r.retract(new Fact("foo", r));
r.retractString("(imagonna)");
      
```


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Getting information from the Rete

- Various methods return iterators
 - `Rete.listFacts();`
 - `Rete.listDefrules();`
 - `Rete.listDeftemplates();`
 - `Rete.listDeffacts();`
 - `Rete.listFunctions();`
- `listFacts()` most useful
 - Can use `toString()`


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Listening to JESS

- Much like listening to AWT/Swing events
- Interface `JessListener`
 - `void eventHappened(JessEvent je)`
 - `r.addJessListener(JessListener jel);`
- `JessEvent`
 - `int getType();` types are defined as public ints
 - `Object getObject();` returns the object corresponding to the event


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Values

- Generic holder for symbols ints reals strings
- Associated with Facts
- May be returned from the `JessListener`
 - `longValue()`
 - `doubleValue()`
 - `stringValue()`
 - `type()` (as an int)
 - `RU.ATOM`, `RU.STRING`, `RU.SLOT`, `RU.MULTISLOT`.


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Advantages of JESS

- Scripting is a Good Thing
 - You don't need to recompile the interface, you can easily fix the inference engine
- Allows a GUI or internet interface for the inference engine
 - (clips console == ugly)
- Can have multiple Rete Classes in a single program
 - Dealing with multiple instances of the same problem (using `Rete.clone()`)
 - Allow concurrent different problems to be run


© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Ideas for using JESS

- Parse a script then assert the descriptive facts, run the engine and get the solution.
 - Note: you need to make sure that the script and the program agree on output
- Rete classes aren't threaded: add a threaded subclass and have them run concurrently
- As mentioned: pretty GUI or server
- Use the java program to redirect output from one inference engine as input to another.

© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz



Recap

- The Rete and Jesp classes
- Adding information to the Rete
- Getting information from the Rete
- Listening to the Rete
- Some tips and ideas

© University of Auckland www.cs.auckland.ac.nz/~ian/ ian@cs.auckland.ac.nz