

The CLIPS Programming Tool

- **History of CLIPS**
 - Influenced by OPS5 and ART
 - Implemented in C for efficiency and portability
 - Developed by NASA, distributed & supported by COSMIC
 - Runs on PC, Mac, also under UNIX and VAX VMS
- **CLIPS provides mechanisms for expert systems**
 - A top-level interpreter
 - Production rule interpreter
 - Object oriented programming language
 - LISP-like procedural language

Components of CLIPS

- **Rule-Based Language**
 - Can create a fact list
 - Can create a rule set
 - An inference engine matches facts against rules
- **Object-Oriented Language**
 - Can define classes
 - Can create different sets of instances
 - Special forms allow you to interface rules and objects

Defining Facts

- **Facts can be asserted**

```
CLIPS> (assert (today is sunday))  
<Fact-0>
```

- **Facts can be listed**

```
CLIPS> (facts)  
f-0 (today is sunday)
```

- **Facts can be retracted**

```
CLIPS> (retract 0)  
CLIPS> (facts)
```

Managing Facts

- **Clearing all facts**

CLIPS> (clear)

CLIPS> (facts)

- **Grouping facts - typically in a file**

(defacts today

; can be cleared with (undeffacts today)

(today is sunday)

(weather is warm)

)

- **After loading facts, assert with (reset)**

Defining Rules

```
(defrule rule-name  
  optional-comment  
  optional-declaration  
  condition ... condition  
  =>  
  action ... action  
)
```

A Sample CLIPS Rule

```
(defrule sunday "Things to do on Sunday"  
  (salience 0)  
  (today is Sunday)  
  (weather is sunny)  
  =>  
  (assert (chore wash car))  
  (assert (chore chop wood))  
)
```

Getting the Rules Started

- The reset command creates a special fact

```
CLIPS> (load "today.clp")
```

```
CLIPS> (facts)
```

```
CLIPS> (reset)
```

```
CLIPS> (facts)
```

```
f-0 (initial-fact) ...
```

```
(defrule start
```

```
  (initial-fact)
```

```
  =>
```

```
  (printout t "hello")
```

```
)
```

Tracing & Recording Things

- **Watch command can watch facts (and rules)**

CLIPS> (watch facts)

CLIPS> (reset)

==> f-0 (initial-fact)

CLIPS> (retract 0)

<== f-0 (initial-fact)

- **Contents of dialog window can be sent to file**

CLIPS> (dribble-on "dribble.clp") ; any file name will do

...

CLIPS> (dribble-off "dribble.clp")

Variables & Pattern Matching

- Variables make rules more applicable

```
(defrule pick-a-chore
  (today is ?day)
  (chore is ?job)
  =>
  (assert (do ?job on ?day))
)
```

- If conditions matched, then bindings are used

Retracting Facts from a Rule

```
(defrule do-a-chore
```

```
  (today is ?day) ; ?day must have a consistent binding
```

```
  ?chore <- (do ?job on ?day)
```

```
  =>
```

```
  (printout t ?job “ done”)
```

```
  (retract ?chore)
```

```
)
```

- Must assign a variable to item for retraction

Pattern Matching Details

- **One-to-one matching**

(do ?job on ?day)

(do washing on monday)

- **Use of wild cards**

(do ? ? monday)

(do ? on ?)

(do ? ? ?day)

(do \$?)

(do \$? monday)

(do ?chore \$?when)

Using Templates

```
(deftemplate student "a student record"  
  (slot name (type STRING))  
  (slot age (type NUMBER) (default 18)))
```

```
CLIPS> (assert (student (name fred)))
```

```
(defrule print-a-student  
  (student (name ?name) (age ?age))  
  =>  
  (printout t name? " is " ?age)  
)
```

Defining Functions in CLIPS

- Uses a LISP or Scheme-like syntax

```
(deffunction function-name (arg ... arg)  
  action ... action)
```

```
(deffunction hypotenuse (?a ?b)  
  (sqrt (+ (* ?a ?a) (* ?b ?b))))
```

```
(deffunction initialize ()  
  (clear)  
  (assert (today is sunday)))
```

Defining Classes & Instances

- Defining the class CAR

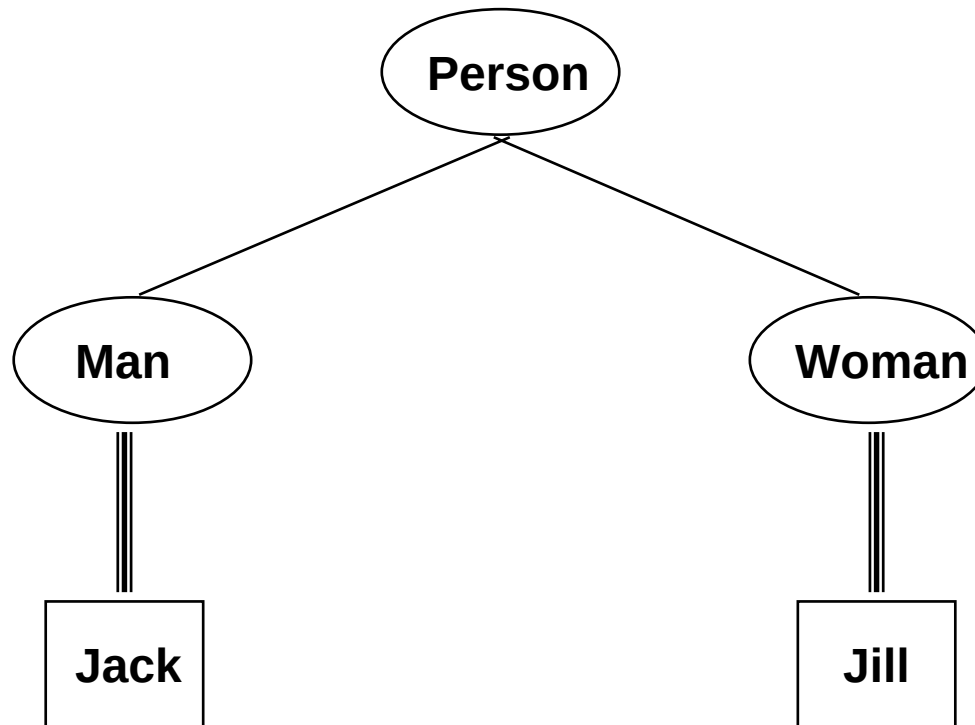
```
(defclass car  
  (is-a user)  
  (name)  
  (made-by))
```

- Defining an instance of CAR

```
(make-instance corvette of car  
  (made-by chevrolet))
```

Concrete & Abstract Classes

- Some classes only exist for inheritance purposes



Managing Instances

- **Commands to display instances**

CLIPS> (instances)

[corvette] of car

CLIPS> (send [corvette] print)

[corvette] of car

(made-by chevrolet)

- **Command to group instances (in a file)**

(definstances

(corvette of car (made-by chevrolet))

(thunderbird of car (made-by ford)))

Clearing & Resetting Instances

- **Deleting an instance**

CLIPS> (send [corvette] delete)

- **Deleting all instances**

CLIPS> (unmake-instance *)

- **Resetting creates an initial object**

CLIPS> (reset)

CLIPS> (instances)

[initial-object] of INITIAL-OBJECT

Message Passing

- The SEND function

`(send [instance] message arg ... arg)`

- Converting from symbols to names

`CLIPS> (symbol-to-instance-name corvette)
[corvette]`

This is useful when SENDing from inside a rule

Limitations of CLIPS

- **Single level rule sets**
 - in LOOPS, you could arrange rule sets in a hierarchy
- **Loose coupling of rules and objects**
 - rules can match on objects and communicate with them via message passing
 - rules cannot easily be embedded in objects, as in Centaur
- **CLIPS has no explicit agenda mechanism**
 - the basic control flow is forward chaining
 - to implement other kinds of reasoning you have to manipulate tokens in working memory

Alternatives to CLIPS

- **Eclipse**
 - has same syntax as CLIPS (both are based on ART)
 - supports goal-driven (i.e., backwards) reasoning
 - has a truth maintenance facility for checking consistency
 - can be integrated with C++ and dBase
 - new extension RETE++ can generate C++ header files
- **NEXPERT OBJECT**
 - another rule- and object-based system
 - has facilities for designing graphical interfaces
 - has a ‘script language’ for designing user front-end
 - written in C, runs on many platforms, highly portable