CS 367 Tutorial
4 August 2008
Week 3 (tutorial #1)
Carl Schultz

CLIPS

Homepage:
http://clipsrules.sourceforge.net/

CLIPS Documentation

Two CLIPS documents: (1) User's Guide (2) Reference Manual

- **User's Guide**
    - Introduction to CLIPS
      http://www.cs.auckland.ac.nz/compsci367s2c/resources/clips/documentation/usrguide.pdf
- **Reference Manual**
    - Volume I: "Basic programming guide"
      http://www.cs.auckland.ac.nz/compsci367s2c/resources/clips/documentation/bpg.pdf
      Syntax definitions and examples
    - Volume II: "Advanced Programming Guide"
      advanced stuff, experienced users
    - Volume III: "Interfaces Guide"
      details about machine-specific interfaces

CLIPS language

facts

Uses **symbols** – a symbol is a sequence of ascii characters (with a few exceptions)

```
> (assert (green frog))
> (facts)
```

NB: all facts are given an unique identifier by CLIPS
e.g. f-1 (green frog)

Use **retract** to remove a fact

```
> (retract 1)
```

templates

```
>   (deftemplate frog "info about a frog"
      (slot name)
      (slot age)
)
> (list-deftemplates)
> (assert (frog (name jane)))
```

When using templates to make facts, don't forget to start with "assert"
e.g. "(assert (frog …"

<u>deffacts</u>

Useful if the same set of assertions will be used every time a program is run

```
> (deffacts nice "stuff that is tasty"
      (nice watermelon)
      (nice fudgecake)
    )
> (list-deffacts)
```

Activate deffacts by **resetting** the facts:

```
> (reset)          …only clears facts (keeps rules and deffacts)
```

<u>deffunction</u>

Functions compute simple values
Many built-in:

```
> (+ 5 2)
> (sin 0.2)
…
```

We can define functions using **deffunction**
Use **variables** – a variable starts with "?", e.g. ?name
When referring to variables inside a function, don't forget to always include "?"

```
> (deffunction increment (?i)
      (+ ?i 1)
    )
> (increment 6)                 …returns "7"
> (list-deffunctions)
> (undeffunction increment)     …remove the function
```

<u>defrule</u>

Rules are: IF conditions THEN (=>) results
Don't forget to "assert" facts on right hand side (after "=>")

```
> (defrule weather
      (or (wearing raincoat) (holding umbrella))
      =>
      (assert (raining))
    )
> (rules)
> (agenda)   …shows which rules are ready to fire
```

Start using rules with **run**, e.g.

```
        > (assert (holding umbrella))
        > (run)
        > (facts)
           …
           (raining)
           …
        > (undefrule weather)
```

…remove the rule

bind

Associate symbols (e.g. “bill”, “<Fact-1>”, “4” etc.) to variables (e.g. “?name”)

```
        > (bind ?percent (random 1 100))
```

Other bits and pieces

To load an example file (e.g. “stove.clp”) use **load** command

```
        > (load “C:/stove.clp”)
        > (reset)
        > (run)
```

The **open** command is used for file i/o (reading / writing)

```
        > (open “mfile.clp” file-handle “r”)
        > (readline file-handle)
        …
```

If you’re using the command prompt and nothing happens when you press enter (just a blank new line) then you might need to add (a) a closing bracket ‘**)**’ or (b) closing quotes ‘ **”** ‘

many other keywords

multislot
allowed-symbols
allowed-numbers
type

debugging

```
        > (watch facts)
        > (watch activation)
        > (unwatch facts)
        > (ppdefrule weather)
```

…shows rule

```
        > (ppdeffunction increment)
```

…shows function

```
        > (ppdeffacts nice)
        > (printout t "quack" crlf))
```

…prints quoted text (t means terminal, crlf means new line)

style conventions

refer “A Matter of Style” pg.24 of the user guide file.