An Introduction to the CLIPS Programming Language

Jack L. Watkin
Department of Electrical and Computer Engineering
University of Dayton
Dayton, Ohio 45469–0232 USA
watkinj1@udayton.edu

ABSTRACT

We provide an introduction to CLIPS—a declarative programming language for implementing expert systems.

KEYWORDS

CLIPS, expert systems, rule-based language.

Jack L. Watkin. 2017. An Introduction to the CLIPS Programming Language. *CPS 499-03: Emerging Languages, University of Dayton, Ohio 45469–0232 USA, Spring 2017,* 4 pages.

1 INTRODUCTION

Originally called NASA's Artificial Intelligence Language (NAIL), CLIPS started as a tool for creating expert systems at NASA in the 1980s. CLIPS stands for **C** **L**anguage **I**ntegrated **P**roduction **S**ystem. In AI, a production system is a computer system which relies on facts and rules to guide its decision making. CLIPS is a nonprocedural, declarative, and rule-based language, which is different than more traditional programming languages like C or LISP. Fig. 1 situates CLIPS in relation to other programming paradigms and languages [2].

2 CLIPS FROM THE COMMAND LINE

The CLIPS shell can be invoked in UNIX-based systems through the `clips` command. From within the shell, the user can `assert` facts, `defrules`, and `(run)` the inference engine. To load facts and rules from an external file, use the `-f` option (e.g., `clips -f database.clp`). Table 1 is a summary of commands to be used from within the CLIPS shell or that can be used in CLIPS scripts.

3 EXPERT SYSTEMS IN CLIPS

An *expert system* is a computer program capable of modeling the knowledge of a human expert [3]. In CLIPS expert systems, knowledge is represented as *facts* and *rules*. *Facts* are axioms that are asserted as true. *Rules* are declarations expressed in the form of an `IF THEN` statement. For example, a fact may be 'It is raining,' and a rule could be 'If it is raining, then I carry an umbrella.' In CLIPS this fact and rule could be written as:

```
(assert (weather raining))
```


CPS 499-03: Emerging Languages, University of Dayton, Ohio 45469–0232 USA

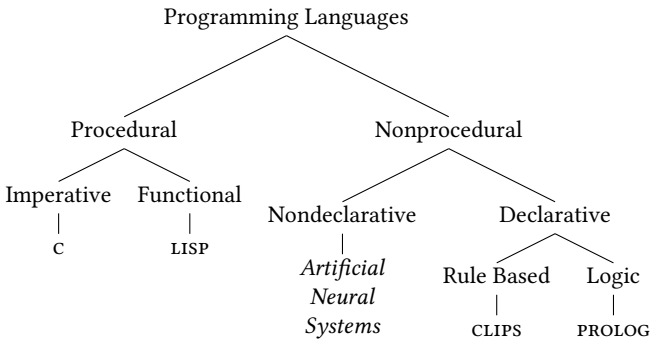


Figure 1: A hierarchy of programming paradigms and languages (adapted from [2]).

Table 1: Essential CLIPS shell commands.

Command	Function
`(run)`	Run the inference engine.
`(facts)`	Retrieve the current fact-list.
`(clear)`	Restores CLIPS to start-up state.
`(retract n)`	Retract fact `n`.
`(retract *)`	Retract all facts.
`(watch facts)`	Observe facts entering or exiting memory.

```
(defrule
    (weather raining)
    =>
    (assert (carry umbrella)))
```

The `assert` keyword defines facts, which are inserted in FIFO order into the `fact-list`. Facts can also be added to the `fact-list` through the `deffacts` command. The following is the general syntax [3][1] of a rule:

```
(defrule rule_name
        (pattern_1)   ; IF Condition 1
        (pattern_2)   ; And Condition 2
        .
        .
        (pattern_N)   ; And Condition N
        =>   ; THEN
        (action_1) ; Perform Action 1
        (action_2) ; And Action 2
```

[1]Note that ; begins a comment.

```
    .
    .
    (action_N)) ; And Action N
```

When the user issues the `(run)` command, the `clips` inference engine pattern matches facts with rules. If all patterns are matched within the rule, then the actions associated with that rule are fired.

4 THEORY OF CLIPS

In logic programming languages like PROLOG, the inference engine is given a goal that it then sets out to prove based on the system's knowledge base. This approach is called *backward chaining* because the system works backward from a goal to find a path to prove that the goal is true. CLIPS works in the opposite direction. CLIPS takes asserted facts and attempts to match them to rules to make inferences. This is known as *forward chaining*. The diagram shown in Fig. 2 illustrates the overall architecture of the CLIPS system [4].

The *Match-Resolve-Act Cycle* is the foundation of the CLIPS inference engine which performs pattern matching between rules and facts through the use of the *Rete Algorithm*. Once the CLIPS inference engine has matched all applicable rules, conflict resolution occurs. Conflict resolution is the process of scheduling rules that were matched at the same time. Once the actions have been performed, the inference engine returns to the pattern matching stage to search for new rules that may be matched as a result of the previous actions. This continues until a fixed point is reached.

5 FEATURES

5.1 Variables

Variables in CLIPS are prefixed with a ? (e.g., ?x). Variables need not be declared explicitly. However, variables must be bound to a value before they are used. Consider the following program that computes a factorial:

```
(defrule factorial
    (factrun ?x)
    =>
    (assert (fact ?x 1)))

(defrule facthelper
    (fact ?x ?y)
    (test (> ?x 0))
    =>
    (assert (fact (- ?x 1) (* ?x ?y))))
```

When the facts for the rule `facthelper` are pattern matched, `?x` and `?y` are bound with values. Next, the bound value for `?x` is used to evaluate the validity of the fact `(test (> ?x 0))`. When variables are bound within a rule, that binding exists only within that rule. For persistent global data, `defglobal` should be used as follows:

```
(defglobal ?*var* = "" )
```

Assignment to global variables is done with the `bind` operator.

5.2 Templates

Templates are used to associate related data (e.g., facts) in a single package—similar to `structs` in C. Templates are containers for multiple facts, where each fact is a `slot` in the template. Rules can be pattern matched to templates based on a subset of a template's slots. Below is a demonstration of the use of pattern matching to select specific data from a database of facts.

```
(deftemplate car
   (slot make
     (type SYMBOL)
     (allowed-symbols
      truck compact)
     (default compact))
   (multislot name
      (type SYMBOL)
      (default ?DERIVE)))
(deffacts cars
   (car (make truck)
        (name Tundra))
   (car (make compact)
        (name Accord))
   (car (make compact)
        (name Passat)))

(defrule compactcar
   (car (make compact)
         (name ?name))
   =>
   (printout t ?name crlf))
```

5.3 Conditional Facts in Rules

Pattern matching need not match an exact pattern. Rather logical operands can be applied to the patterns to support conditional matches. This is done through the or (|), and (&), and not (~) operators. The following rule demonstrates the use of these operators:

```
(defrule walk
   (light ~red&~yellow) ;if the light
                        ;isn't yellow and
                        ;isn't red
   (cars none|stopped)  ;no cars or stopped
   =>
   (printout t "Walk" crlf))
```

6 DECISION TREES, FLOW CHARTS, AND STATE MACHINES

An application of CLIPS is decision trees. More generally, CLIPS can be applied to graphs that represent a human decision-making process. Facts can be thought of as the edges of these graphs, while rules can be thought of as the actions or states associated with each node of the graph.

One example of this decision making process is an expert system that emulates a doctor in treating, diagnosis, and explaining diabetes [1]. The patient asserts facts about herself including eating habits, blood-sugar levels, and symptoms. The rules within this expert system match these facts and provide recommendations about

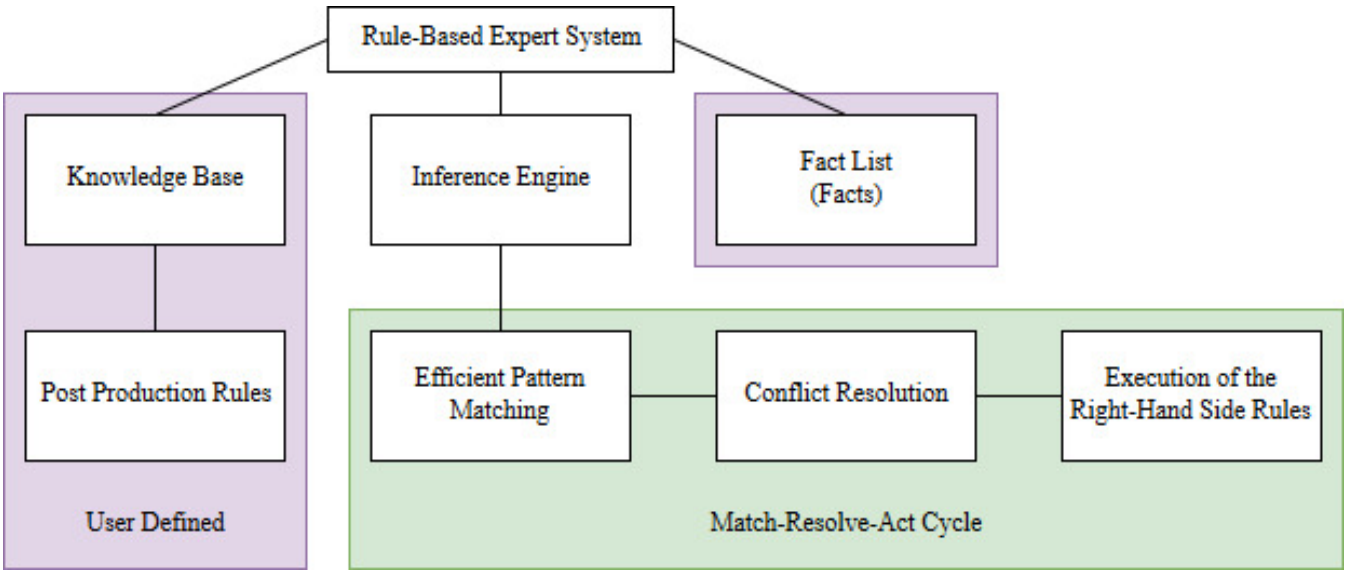


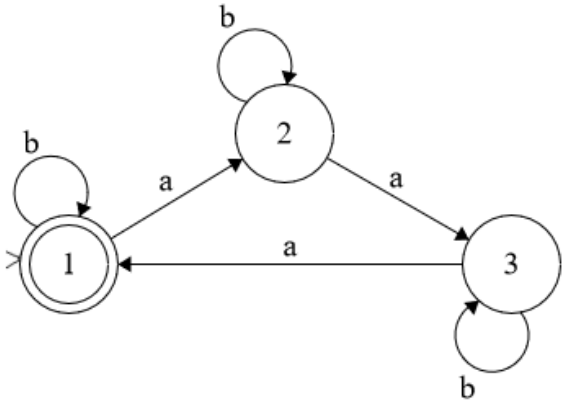
Figure 2: Overall architectural design of CLIPS (adapted from [2]).

managing diabetes in the same way a doctor may interact with a patient.

7 EXERCISES

The following are programming exercises that incorporate essential CLIPS concepts:

(1) Build a state machine using CLIPS that accepts a language L consisting of strings in which the number of a's in the string is a multiple of three over an alphabet {a,b}. Hint: use the following state machine for L:



Examples:

```
CLIPS> (run)
String? aaabba
Rejected

CLIPS> (reset)
CLIPS> (run)
String? aabbba
Accepted

CLIPS>
```

(2) Rewrite the factorial program in § 5.1 so that only the fact with the final result of the factorial rule is stored in the fact list. Hint: `retract` can be used to remove facts from the fact list.
Examples:

```
CLIPS> (assert (fact_run 5))
CLIPS> (run)
CLIPS> (facts)
f-0    (fact_run 5)
f-1    (fact 0 120)

CLIPS>
```

8 CONCLUSION

CLIPS is a language which allows for the implementation of 'rule of thumb' knowledge similar to that of a human expert. These forward-chaining systems help a computer make heuristic decisions by considering the facts (or lack of facts) of a situation and drawing conclusions. Because CLIPS can handle arbitrarily large data, it has an advantage over human experts because humans are limited in their ability to consider many facts at once.

REFERENCES

[1] M. Garcia, T. Gandhi, J. Singh, L. Duarte, R. Shen, M. Dantu, S. Ponder, and H. Ramirez. 2001. *Esdiabetes (an Expert System in Diabetes).* Consortium for Computing Sciences in Colleges. 166–175 pages.

[2] J.C. Giarratano and G. Riley. 1998. *Expert systems principles and programming.* PWS Publishing Company, Boston, MA.
[3] J. C. Giarratano. 2008. *CLIPS User's Guide.* The MIT Press, Cambridge, MA.
[4] P. Lucas and L. van der Gaag. 1991. *Principles of expert systems.* Addison-Wesley, Boston, MA.