N88-16365

CLIPS AS A KNOWLEDGE BASED LANGUAGE

JAMES B. HARRINGTON
HONEYWELL SPACE AND STRATEGIC AVIONICS DIVISION
CLEARWATER FL 34624-7920

ABSTRACT

CLIPS is a language developed by Johnson Space Center (JSC) for writing expert systems applications on a personal or small computer. The CLIPS language was written in the C programming language and JSC made provisions to call CLIPS from, or embed CLIPS within, a control or applications program. This paper will look at some of the salient characteristics of a knowledge based system (KBS). The capabilities of CLIPS will be discussed in light of these characteristics, and the KBS characteristics of CLIPS will be compared with those of LISP, Prolog, and OPS5.

INTRODUCTION

The intent of this paper is to describe the CLIPS programming language and compare it to three other artificial intelligence (AI) languages (LISP, Prolog, and OPS5) with regard to the processing they provide for the implementation of a KBS. The paper will conclude with a discretion of how CLIPS would be used in a control system. The definition of many of the commonly used terms in the field of AI languages will be found in this paper.

PROGRAMMING LANGUAGES

Several languages have been developed to enhance the building of KBS by providing a direct method of encoding both data and procedural knowledge (procedural knowledge is the knowledge of how to act on the data). The major requirement for a language to be used for developing a KBS is that it handle strings of characters or "symbols" as well as numbers. For the above reason Pascal and C are more favored, among the "standard" programming languages, for developing expert systems than is FORTRAN or assembly language. Several languages have been developed specifically to enhance the capability to deal with symbols; of these languages, this paper will deal with only LISP, Prolog, OPS5, and CLIPS.

The most common language for developing AI applications is LISP. LISP stands for *LIS*t *P*rocessing language. It was based on John McCarthy's work on nonnumeric computation published in 1960. LISP itself does not have any constructs that provide for explicit encoding of data and procedural knowledge, however, LISP is an excellent symbol processing language and provides a rich set of tools that can be used to develop the constructs desirable for a KBS.

Prolog is a relatively new language that has been developed for AI applications. Prolog stands for *Pro*gramming in *Log*ic. It was one of the first attempts to structure a language that would enable a programmer to specify his tasks in logic rather than in conventional programming methods. Prolog was created by Alain Colmerauer and his associates at around 1970.

The name OPS5 stands for *O*fficial *P*roduction *S*ystem, version 5. As one might expect, OPS5 grew out of set of OPS languages. The pilot system developed in OPS5 was "R1" for Digital Equipment Corporation (for the VAX Expert System (ES) configuration tool). C. Forgy and J. McDermott, of Carnegie-Mellon University, were responsible for the development of the OPS5 language.

CLIPS is the most recently developed language of the set to be discussed in this paper. CLIPS is a Forward Chaining rule based system. It is being developed by the Johnson Space Center's AI section, Mission Planning and Analysis Division, as a language suitable for ES development and delivery on conventional computers (ie. the IBM PC, VAX, etc.) and is intended for embedded applications. CLIPS was originally created by Frank Lopez around 1985 and reworked for release to the public by Gary Riley [Culbert 86]. CLIPS is an acronym for *C* *L*anguage *I*ntegrated *P*roduction *S*ystem.

KBS CONCEPTS

A KBS is a program or system that uses a base of knowledge to determine the program output. A KBS language is one that enhances the capabilities of combining data and production rules to obtain a meaningful output. The following sections describe some of the main concepts or characteristics of a KBS.

KBS vs Conventional Languages

A KBS language could be described as a language based on a set of rules that act like functions in a conventional language. These rules are triggered by data (or facts) rather than program flow. All of the facts are examined by the rules on a continuous basis. Hence the KBS code need not execute in the logical flow that it was written. There are often mechanisms for controlling the flow of rule activation (executing a given rule in a KBS) but in general, if the order that decisions are made can be predetermined, and remain constant regardless of the data, then a conventional programming languages would be a more appropriate selection.

Another important difference in the AI languages and conventional programming languages is the way variables a handled. In the AI languages the variable only has meaning within the particular rule in which it is located, there are no global variables. The only method available for "passing parameters" is by asserting a new fact on the fact list.

AI Facts

A fact can be a single element or a list of elements. Each indivisible element in a fact is called an "atom". One of the main reasons that LISP has became so popular for AI applications is because of its built in capability to work with lists.

Table 1 is a summary of the capabilities of each language to represent facts in the knowledge base. The "Argument Format" column specifies weather the element's value is based on its position in the fact list (positional) or is based on keyword recognition. The "predicate" column refers to the association of an atom within the fact to a header or a name; CLIPS is the only language that does not directly provide the capability of relating atoms to a name or function, however, the programmer can define a structure where certain positions within a fact are keywords and the other positions are variable values.

AI Rules

The executable "code" in a KBS are rules. A rule can be viewed as a special If/Then statement and can be partitioned into two parts. The if-part or logic section of the rule is the part of the rule that looks for matches and relationships among the data. The then-part or action part of the rule is activated only after the conditions in the if-part have been satisfied.

Table 1: Language Implementation of Facts*

Language	Argument Format	Predicate	Argument Name	Argument Value
LISP (list)	positional	first list element	n/a	rest of list
LISP (structure)	keyword	type name	slot name	slot value
Prolog	positional	function	n/a	argument
OPS5	keyword	class name	attribute	value†
CLIPS	positional	user-defined	user defined	value†

† must be atomic

LISP does not have any constructs that directly implement an If/Then type of statement. However, LISP does have the language constructs to build If/Then type rules that could allow multiple patterns to be matched as well as multiple actions to be performed. Prolog, OPS5, and CLIPS each provide for multiple pattern, pattern matching capabilities which are summarized in Table 2. In Table 2, "Conjunction" refers to the logical ANDing of facts. "Disjunction" is the logical ORing of facts. Table 3 is a table of commonly used knowledge base operations.

Table 2: If-part Pattern Matching**

Language Feature	Prolog	OPS5	CLIPS
=, ≠ (equal, not equal)	any term	number, symbol, predicate	number, symbol, function
<, >, . . .(less than, greater than . . .)	any term	number	number
Computed expressions	Yes (if-part only)	No (then-part only)	Yes (both parts)
Type test	atom, number, variable	number, symbol	atom, number
Negation, Conjunction	predicate, argument	predicate, argument	argument, function
Disjunction	predicate, argument	argument	argument, function
Nesting of Conditions	Yes	No	Yes

File I/O

A key part of the KB operations is the capability for interfacing with a "permanent" data base. A permanent data base is usually stored on magnetic disk, thus the capability to interface with a permanent data base relies on file I/O capabilities. Of the four languages, LISP has the most extensive set of I/O capabilities; OPS5 has the smallest set.

CLIPS has two sets of I/O file commands: the first is for saving and retrieving program files; the second is for saving and retrieving facts. The CLIPS facilities for saving rules from the CLIPS environment will save only the rules; there is no top

*Expanded from [Cugini 87], Table 3, p. 20.
**Expanded from [Cugini 87], Table 4, p. 23.

level command (like SAVE) to save the facts in the fact list or any "deffacts" statements (deffacts is a CLIPS construct that allows the user to develop a set of facts). CLIPS will load both rules and facts if the program file is created with an external text editor and the deffacts construct is used. During CLIPS program execution, CLIPS does support reading facts from, and writing facts to, disk files.

Table 3: Operations Of Rules On Facts*

Operation	Number of KB objects	Type of KB objects	Source	Language Statement	Rule-part containing the Statement
LISP:					
add	many	fact, rule	file	load	user-defined
add	one	fact, rule	program	make-x	user-defined
modify	one	fact, rule	program	setf	user-defined
delete	one	fact, rule	program	remove,	user-defined
		fact, rule		remhash,...	user-defined
Prolog:					
add	one	fact	program	implicit†	then
add	one	fact, rule	program	assert	if
delete	one	fact, rule	program	retract	if
delete	many	fact, rule	program	abolish	if
add	many	fact, rule	file	consult	if
replace	many	fact, rule	file	reconsult	if
OPS5					
add	one	fact	program	make	then
modify	one	fact	program	modify	then
delete	one	fact	program	remove	then
add	one	fact	program	build	then
CLIPS:					
add	many	fact	file	read <file>	then
add	many	fact	keyboard	read	then
add	many	fact	program	assert	then
delete	many	fact	program	retrace	then

† does not persist--derived and then discarded.

Table 4 is a summary of the I/O features of the four languages. "I/O language objects" refers to the ability to read objects as elements; each read associates a variable with an atom. "I/O characters" refers to the capability to read the external file a character at a time. "I/O binary" refers to the capability of treating an input as a set of bits where each bit may have a specific meaning. The user can modify the CLIPS source code to provide both character and binary input capabilities. "Line input" is the capability to input a line of data at a time regardless of the number of elements associated with the data line. "Pseudo-I/O" is the capability to treat internal memory as an I/O buffer and manipulate memory using I/O routines. "User control" of the input and output refers to the facilities the user has to control the format of the inputting and outputting of data. "Rename and delete files" refers to the capability to access system file commands form the language environment. The

*Expanded from [Cugini 87], Table 5, p. 25.

newer versions of CLIPS do provide access to the DOS commands but access is system dependent.

Table 4: Files and I/O Features*

Features	LISP	Prolog	OPS5	CLIPS
File types	Sequential and Random	Sequential	Sequential	Sequential
I/O language objects	Yes	Yes	Yes	Yes
I/O characters	Yes	Yes		User enhancable
I/O binary	Yes			User enhancable
Line input	as characters		as language objects	as language objects
Pseudo-I/O	Yes			
User control of input	Many Features	Some		Few
User control of output	Many Features	Some	Few	Many Features
Rename and delete files	Yes	Yes		New Versions

Inference Engines

The inference engine is the part of the language that derives the response to a set of facts. It is responsible for selecting which rules will be fired in which order.

The top level of the inference engine is how (or in what order) the facts are processed. Two of the most common terms used to describe the control strategy of rule firing are forward chaining and backward chaining. Forward chaining starts with a set of facts and processes facts with rules until it has reached some conclusion or until there are no more facts to process. Backward chaining starts with a conclusion or assertion of a condition and then checks the facts to determine if the condition can be supported. Backward chaining is extremely useful in any system where the program may be asked why it chose a specific course of action.

Another consideration for control strategy is the selection (or decision) of which rule to fire next. Rule activation relies on "control knowledge." The relationship between rules and control knowledge is that, "rules capture knowledge about *how to transform data*; control knowledge is about *when to transform data*" ([Cugini 87] p. 15, my italics). The commonly used terms for the decision making process are: Depth-first, Breadth-first, Recent-first, Best-first, and Heuristic [Cugini 87].

Prolog is a backward chaining system that processes facts in a depth-first fashion. A depth-first system tries to process as far down one decision path as possible until a block is encountered (in the form of an unsupported fact or improper conclusion). When a block is reached, a depth-first system will back up to the last successful node and proceed down the next alternate path. This process continues from left to right across the "decision tree" until the solution is found or all decision paths are exhausted.

Both OPS5 and CLIPS use forward chaining systems with recent-first fact processing. The recent-first technique does not necessarily progress toward an answer. In a recent-first system, the most recently asserted facts are given more weight so that rules using these facts would be fired first (unless there is some other weighting

*Expanded form [Cugini 87], Table 8, p. 52.

system which might override the rule firing order). Both OPS5 and CLIPS will continue to process facts until each rule has processed each applicable fact.

It is important to note that just because a language was designed around a specific type of inference engine, that language is not locked into that role. There are many cases where Prolog has been used to implement a forward chaining system. Likewise, OPS5 and CLIPS have been used to create backward chaining systems.

User Interface

Any good language will provide tools to aid in debugging of the source code. The most often used tools are: trace features (which will indicate which line of code is being executed), break points (where the number of lines to be executed is specified or an event is specified which will stop execution), and printing out changes in the values of variables. In some languages these tools must be written as part of the source code.

Table 5 is a summary of the user interface features that might be used for debugging a KBS. "Top-level control" refers to the process of invoking or running the KBS. The section on "watch derivation" refers to watching the "thinking" process of the KBS as it moves toward its end point. The "pause and step" features refer to the KBS executing a set number of cycles or instructions. Either before the KBS is executed or during a pause in the execution of a KBS a user may want to "inspect the KBS" (it's facts, it's rules, and the agenda--also known as the conflict list--for rules pending execution). "Manipulate KB" refers to the process of adding, or deleting, facts and rules during a pause in the KBS execution. "Manipulate derivation" refers to controlling the KBS during execution.

Embeddability

Of the four languages discussed, CLIPS is the only language that was designed to be embedded within another system. When CLIPS is purchased, the C source code is also supplied. The CLIPS User's Guide, Reference Manual, and Update notices supply information for customizing CLIPS and embedding CLIPS within other systems. The instructions are written around the Latice C compiler for the IBM PC however there is some information related to using the Lightspeed C compiler on the Macintosh.

Miscellaneous Language Features

CLIPS provides language constructs to perform algorithmic types of tasks. These constructs include If/Then/Else, and Do-While statements which can be executed in the then-part of the rule. Another feature that is useful in CLIPS is the ability to assign "weights" (call salience values) to rules. The rule with the highest salience value is the rule that will fire next. Once all of the criteria are met to satisfy the if-part of the rule the then part of the rule can then assert or retract facts required to control the flow to the next rule to be fired. The process of controlling some of the flow of rule firing, and the use of algorithmic constructs within a rule, greatly enhances CLIPS capability to perform systems simulations as well as making it easier for a conventional programmer to understand some of what is happening within the CLIPS program.

In addition to the language constructs provided, the user may also customize CLIPS for a particular task. The CLIPS User's Guide provides an example for adding a random number generator to CLIPS. The process shown in the User's Guide will work for any function associated with the then-part of the rule. The user could add functions to convert an atom into a set of characters, or to read binary input from a data file, or developing drivers for special equipment (ie. software drivers for turning on and off solenoids). The capability of customizing CLIPS is an important strength to the language.

Table 5: User Interface Features*

Language Feature	LISP	Prolog	OPS5	CLIPS
Top-level control:				
invoke derivation	Yes	Yes	Yes	Yes
exit derivation		Yes		
exit system	undefined	Yes	Yes	Yes
Watch derivation:				
complete	Yes	Yes	Yes	
selective	Yes	Yes		Yes
Pause and step:				
pause from program	Yes	Yes	Yes	
pause at named entity		Fact or rule	Rule	
pause after n cycles			Yes	Yes
asynchronous interrupt		Yes		
step thru named entity	Yes, via statement	Yes		
step thru all	Yes	Yes	Yes, via run	Yes, via (run 1)
Inspect KBS				
named KB object	Yes	Yes	Yes	Yes, facts as a list rules by name.
matching KB object		Yes	Fact only	
derivation-state		Goal stack	Conflict set	Conflict set
Manipulate KB:				
add KB object		Yes	Facts only	Facts only
delete KB object		Yes	Yes	Yes
Manipulate derivation:				
abort		Yes		System dependent
backup program cycles		Yes	Yes	
continue	Yes	Yes	Yes	Yes (from program errors)
suspend derivation		Yes		Yes (on program errors)

A CLIPS APPLICATION

The CLIPS language is a good choice for writing a control system or simulation of a control system. The rule based nature of the CLIPS language provides an intuitive and quick medium for developing control rules.

As an example, a system designer required that a pressure of a vessel should never exceed 95 psi, and that at 75 psi a warning message should be sent to the operator. The types of rule that would be used to realize this control would be:

*Expanded from [Cugini 87], Table 6, p. 30.

```
(define rule: Warning-message
    if (vessel-pressure => 75)   => (then) (printout "Warning--vessel pressure has
reached " (vessel-pressure/95)*100 " percent capacity))

(define rule: Activate-pressure-relief-valve
    if (vessel-pressure => 95) => (then) (open (pressure-valve-3)) and (printout
"Warning--vessel pressure critical.  Relief valve has been activated"))
```

These rules are not in the CLIPS rule format because the language syntax would look confusing without sufficient explanation. The rule structure is similar though.

Though this in not AI in the strict sense, the rules are capturing the "rules of thumb" that the expert (the system designer) would use to control the system. The real strength of using CLIPS for the control language is that each rule stands on its own, any modification to the system would occur on a rule to rule basis with a minimal to other rules (ie. changing the name of a variable in one rule will have no effect on the function of another rule). For these reasons, Honeywell is reviewing CLIPS as a candidate language for demonstrating embedded ES capability in controllers for the Space Station.

CONCLUSIONS ABOUT CLIPS

Of the four languages, LISP is the most flexible but requires the most work to produce an ES. If the KBS requires high levels of flexibility or different types of inference operations during a single session then LISP would be the better choice of languages. Prolog and OPS5 provide a faster route for developing an ES, while also being easier to maintain, but at the expense of execution time and system memory.

Because of its embedability, its expandability, and its smaller size, CLIPS would be the better selection for embedding low-level ES capability within a control system. CLIPS is similar to OPS5 in its general operation. CLIPS is ment for use on personal computers or smaller computer systems and is the only language that was developed for embedded applications (putting ES capability into another system). Control systems inherently require forward chaining data processing to move from sensor inputs to a controlled output. The forward chaining rule base characteristics of CLIPS make it a good language for developing control systems.