STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

File Copy

February 1970

A LISP IMPLEMENTATION OF BIP

by

Richard E. Fikes

Artificial Intelligence Group

Technical Note 22

INTRODUCTION

This document describes a LISP implementation of BIP (Basic Interface Package) on the PDP-10 computer. BIP is a set of programs designed by Allen Newell at Carnegie-Mellon University which provides the builder of large programming systems a capability for easily defining notational conventions to be used for interacting with a system.* The central routine in BIP is a translator which provides a symbol table and precedence-parsing facility. The entire package provides the following capabilities:

(1) Segmentation of an input stream of characters into "words"

(2) Association of a word to a particular internal symbol

(3) Recognition that some program (action) should be executed upon encountering a particular word in the input

(4) Retention of several symbols and their order of appearance as a context for an action

(5) Declaration of new words and the symbols associated with them; also, declaration of the associated actions, if any

(6) Delay of actions from the time at which their words appear in the input stream until some later time

(7) Association of an internal symbol to an external word

(8) Variation of the symbols and actions associated with a word as a function of context.†

---

*I am indebted to Allen Newell and Peter Freeman for introducing and familiarizing me with BIP. Also, I wish to thank Robert Yates for assisting me with the LISP implementation.

†This list of capabilities is taken from a working paper entitled "BIP: Basic Interface Package" by Allen Newell and Peter Freeman.

BIP was designed to be a skeleton which can be fleshed out in whatever way is useful for the user. The skeleton itself is completely accessible and is meant to be changed to meet the needs of the individual user.

OVERVIEW

In normal usage the BIP translator will remain in top-level control of the user's system throughout a run. The translator uses an EPAM-type discrimination tree to associate actions and internal symbols with strings of characters from the input stream. These associations are made relative to a syntactic and semantic context. The use of contexts provides an extra dimension of flexibility since the user can easily create new contexts, and change contexts during input to allow the interpretation of any given character string to vary depending upon the environment in which it occurs.

The following definitions will help establish a terminology for our further descriptions:

Character--any character which can be input from a teletype.

Word--a string of characters.

Symbol--the internal data structure associated with a particular word. In the SRI BIP the translator calls the function BIP:CRSYM to create a symbol for a new word. At the time of the call, CHARSK is a list (in reverse order) of the characters which make up the word. The symbol created by the BIP:CRSYM function provided with the package is the atom whose name is the same as REVERSE of CHARSK.

Context--a data structure consisting of any or all of the following: a recognition tree, context mark, action list, and boundary character list. In the SRI BIP a context is a list whose first element is the identifier CONTEXT; the recognition tree is an element of the list whose CAR is the identifier TREE; the context mark is the CDR of an element whose CAR is the identifier CM; the action list is an element whose CAR is the identifier ACTIONS; and the boundary character list is an element whose CAR is the identifier BC.

Boundary character--any character used by BIP in determining the boundaries of a word.

Boundary list--part of a BIP context; it is a list of all boundary characters for a particular context.

Action--a BIP data structure which is associated with a symbol and consists of a priority number, an immediate action, and a delayed action. In the SRI BIP an action is a list whose first element is an integer (i.e. the priority number), optional second element is the immediate action, and optional third element is the delayed action. The immediate and delayed actions may be arbitrary evaluable LISP s-expressions.

Action list--part of a BIP context; it is a set of property-value pairs in which the properties are symbols and the values are the actions associated with them. In the SRI BIP an action list is a list whose first element is the identifier ACTIONS and each succeeding element is a list whose CAR is the symbol and whose CDR is the action associated with the symbol.

Context mark--part of a BIP context; it is used to link BIP symbols with nodes of the recognition tree.

Recognition tree--part of a BIP context; it is a discrimination tree used by the translator for the storage of symbol-definition information. In the SRI BIP each node of a recognition tree is a list whose first element is a one-character identifier (except for the top node which has the identifier TREE as its first element) and whose succeeding elements include the nodes which branch from the node and elements whose CAR is the context mark of some context and whose CDR is a BIP symbol.

Data stack--a push-down stack on which a symbol without an action is pushed after its associated word is recognized in the input stream by the translator. In the SRI BIP the data stack is the list DATASK; CAR of DATASK is considered the top element in the stack, CADR of DATASK is the second element, etc.

Operator stack--a push-down stack on which actions containing delayed actions are pushed to await execution of the delayed actions. In the SRI BIP the operator stack is the list OPERSK; CAR of OPERSK is considered the top element in the stack, CADR of OPERSK is the second element, etc.

Context stack--a stack containing pointers to contexts whose top element is the current context. When the translator enters a context it does so by pushing the context being entered onto the context stack. When the translator returns to a previous context it does so by popping the context stack until the desired context is the top element. In the SRI BIP the context stack is the list CONTEXTSK; CAR of CONTEXTSK is considered the top element in the stack, CADR of CONTEXTSK is the second element, etc.

THE TRANSLATOR

The translator's flow of control is shown in Figure 1. The input to BIP is a string of characters from some source such as a teletype, external file, or an internal generator. The translator always calls BIP:GETCHAR to get the next character so that it is independent of the source of these characters and the source can be simply switched. The translator leaves to the user the responsibility of selecting the input source.

The recognition philosophy of BIP is to always recognize the longest possible word. Thus, starting at the top of the tree just after a word has been recognized, BIP will work its way down the branches of the tree as long as possible without checking if the character it has just received is a boundary character or not. When it falls out of the tree, that is, cannot find a branch from the current node labeled with the character that it has just received, it checks to see if the current node contains a context mark identical to that of the current context. If not, or if the current character is not a boundary character, it assumes a new word is being defined and proceeds to extend the tree so that it can now recognize it. If there is a context mark and the current character is a boundary character or the previous character was one, then it knows it has recognized a word and obtains its symbol. If the symbol has an associated syntax action in the current context, it is performed as described below. If it does not have an action, the symbol is pushed onto the data stack. In either case, BIP then begins trying to recognize another word at the point at which the previous word terminates.

In extending the tree to recognize a new word, BIP simply continues to accept new characters until it receives a boundary character. For each new character it adds a new node as a branch from the previous node. When a boundary character is reached, a new data structure (the symbol) is created to associate with the word and a pointer to this structure is stored at the terminal node along with the current context mark; the pointer is stacked on the data stack; and BIP begins trying to recognize another word starting with the boundary character that terminated the new word.

Note that because of the recognition philosophy of BIP it is necessary to have a "quotes context" available to permit the definition of symbols that contain substrings that are symbols and that include a boundary character (once defined, the recognition philosophy permits them to be recognized without any special considerations). For example, we may wish to define the symbols * and *A where * is a boundary symbol. Such a context is supplied as part of the SRI BIP; it has only one boundary character, namely ", and only one syntax action (which is associated with the quote symbol and returns BIP to the previous context). In the above example, suppose we have previously defined * and A as symbols so that they are also boundary characters, and that the action for " in the current context causes the quotes context to be entered.

Then we would write "*A" to define the new word; thereafter, *, A, and *A would be recognized as distinct words.

The recognition and definition of words are lexical actions that are performed by BIP. A user may specify that within any particular context every time a designated word has been recognized a certain syntax action should be taken. This syntax action can be evaluation of an arbitrary function that has been supplied by the user and defined as an action associated with the symbol in the current context. The execution of the action is based on a priority scheme as shown in the flow chart and consists of the execution of an immediate action and possibly an arbitrary number of delayed actions from the operator stack or from the current action (in the order indicated in the flow chart). Since any action (immediate or delayed) is a program, it may do any amount of processing desired; it may work on any of its own data structures or any of BIP's structures (thus effecting BIP's operation) and call any routines whatsoever as subroutines, including the BIP translator itself. In particular, an action may access and alter the data stack (i.e. DATASK) so that the translator acts like a one-stack precedence parser. When the action program is finished, it returns control to BIP which then continues recognizing words in the input stream.

The SRI BIP translator can operate in or out of definition mode. When definition mode is on, all new words are entered into the recognition tree. When it is off, new words are not entered into the recognition tree. A typical use of the mode switch would be to have it on when actions are being defined for key words (e.g. begin, end, if, then) and then turn the switch off when the only new words being encountered are identifiers and numbers. Since the standard BIP:CRSYM will always return the same symbol name for a given word (i.e. the atom whose name is the same as the word), then it is unnecessary and wasteful to have these words in the recognition tree. Definition mode is defined by the value of identifier DEFSWITCH; T denotes definition mode on, NIL denotes off. The translator initializes DEFSWITCH to T.

Note that any one character word which is entered into the tree is also added to the boundary-character list. This is the only built-in mechanism for defining new boundary characters.

If evaluation of an immediate or delayed action causes the value of BIP:RETURN to be set to T, then the translator will return to LISP with a value of NIL immediately following evaluation of the action. This is the only exit mechanism provided in BIP.

INITIAL CONTEXTS

A base context is provided in SRI BIP which includes the necessary facilities for the user to define the language he wishes BIP to read. When the translator is called, this base context (called BIP:BASECON) is made the current context. In normal BIP usage new contexts are created as copies of existing contexts and then built up incrementally; hence all of a user's contexts can have the facilities included in the base context. BIP:BASECON is defined as follows:

Context mark: MARK

Boundary characters: ⟨blank⟩⟨carriage return⟩⟨line feed⟩ " ' ; . ↑

Actions:

⟨blank⟩ ⟨carriage return⟩ ⟨line feed⟩	Read to the next character which is not ⟨blank⟩, ⟨carriage return⟩, or ⟨line feed⟩.
;	Read to the character following the next line feed. Note, this allows comments to be placed on an input line following a semicolon.
"	Enter the quotes context. The quotes context allows the definition of words containing boundary characters (see the discussion above in the section describing the translator and the description below of the quotes context).
'	Use the READ function to read a LISP s-expression and push a pointer to the expression onto the data stack.
. .	The LISP s-expression named in the top of the data stack is popped off the stack and then evaluated using EVAL.
↑	Exit from BIP with the value NIL.

The quote context (named BIP:QUOCON) referred to above in the description of the translator and in the base context's action for double quote is defined as follows:

Boundary characters: "

Actions: " - return to the previous context.

AUXILIARY FUNCTIONS

The following functions are currently defined in SRI BIP:

BIP:ENCON--a MACRO which takes a context pointer as an argument. The context is pushed onto the translator's context stack and made the current context.

BIP:DEFACT--an EXPR taking no arguments which defines the second element in the data stack as the action for the symbol which is pointed to by the top element in the data stack and does two pop operations on the data stack. The definition is made for the current context. For example, the action for the character ↑ in the context BIP:BASECON could be defined as follows:

```
'(100 (SETQ BIP:RETURN T)) ↑ '(BIP:DEFACT)..
```

BIP:CRECON--an EXPR taking no arguments whose value is a newly created context which has the same recognition tree and context mark as the current context and a boundary-character list and actions list which are copies of those of the current context. The user may wish to write other context-creating functions which give the new context a different context mark, a copy of the current context's recognition tree, etc. BIP:CRECON is the only context-creating function provided in SRI BIP.

BIP:DEFCURCON--an EXPR taking no arguments which makes the top context in the context stack the current context.

BIP:RETCON--an EXPR which takes either a positive integer or a context name as an argument. If the argument is an integer k, then the context stack is popped k times; if the argument is a context name, then the context stack is popped until the named context becomes the top element of the stack. After the popping operations are completed, the top element in the context stack is made the current context.

BIP:SKTOP--a MACRO taking the name of a stack as its argument and returning as its value the top element in that stack.

BIP:SKPOP--an FEXPR taking the name of a stack as its argument which pops the stack and returns as its value the element which was popped off the stack.

BIP:SKPUSH--a MACRO which takes a pointer and a stack name as arguments and adds the pointer to the top of the stack. The value of BIP:SKPUSH is a pointer to the resulting stack.

EXAMPLE

To illustrate the use of BIP we present a set of action definitions which will transform algebraic infix expressions into equivalent LISP s-expressions; e.g. A + B will be transformed into (*PLUS A B). The following are examples from the class of expressions to be translated:

A+B+C

(A+B)*C

A+B/-C .

Assuming that LISP has been entered and that the BIP functions have been loaded, the following input sequence will make the desired definitions in a newly created context named INFIX.

```
*(DF DEFBINEXP (L) (BIP:SKPUSH (CONS (CAR L) (REVERSE (LIST (BIP:SKPOP
* DATASK) (BIP:SKPOP DATASK)))) DATASK))

 (DEFBINEXP)

*(BIP)
*'(BIP:ENCON (SETQ INFIX (BIP:CRECON)))..; DEFINE AND ENTER CONTEXT INFIX
*'(6 NIL (DEFBINEXP *PLUS)) + '(BIP:DEFACT)..; DEFINE THE ACTION FOR +
*'(4 NIL (DEFBINEXP *TIMES)) * '(BIP:DEFACT)..; DEFINE THE ACTION FOR *
*'(4 NIL (DEFBINEXP *QUO)) / '(BIP:DEFACT)..; DEFINE THE ACTION FOR /
*'(2 NIL (BIP:SKPUSH (LIST @MINUS (BIP:SKPOP DATASK)) DATASK)) -
*'(BIP:DEFACT)..; DEFINE THE ACTION FOR -
*'(0 (BIP:SKPUSH @(8) OPERSK)) ( '(BIP:DEFACT)..; DEFINE THE ACTION FOR (
*'(8) ) '(BIP:DEFACT)..; DEFINE THE ACTION FOR )
*'(SETQ DEFSWITCH NIL)..; TURN OFF DEFINITION MODE
```

The function DEFBINEXP creates an s-expression to represent a binary algebraic expression. The argument to DEFBINEXP specifies the first element of the created s-expression (the operator), and the top two elements on the data stack specify the second and third elements of the s-expression (the operands). The resulting s-expression is pushed onto the data stack.

The priorities associated with each action provide the desired operator hierarchy. The immediate action for '(' pushes onto the operator stack an action with a lower priority than for any of the operators; the action for ')' is NIL, but its low priority will cause the execution of all delayed actions up to and including the one put into the operator stack by the most recent '('.

Any problems or questions should be directed to Richard Fikes, Room K2090, Extension 4620.

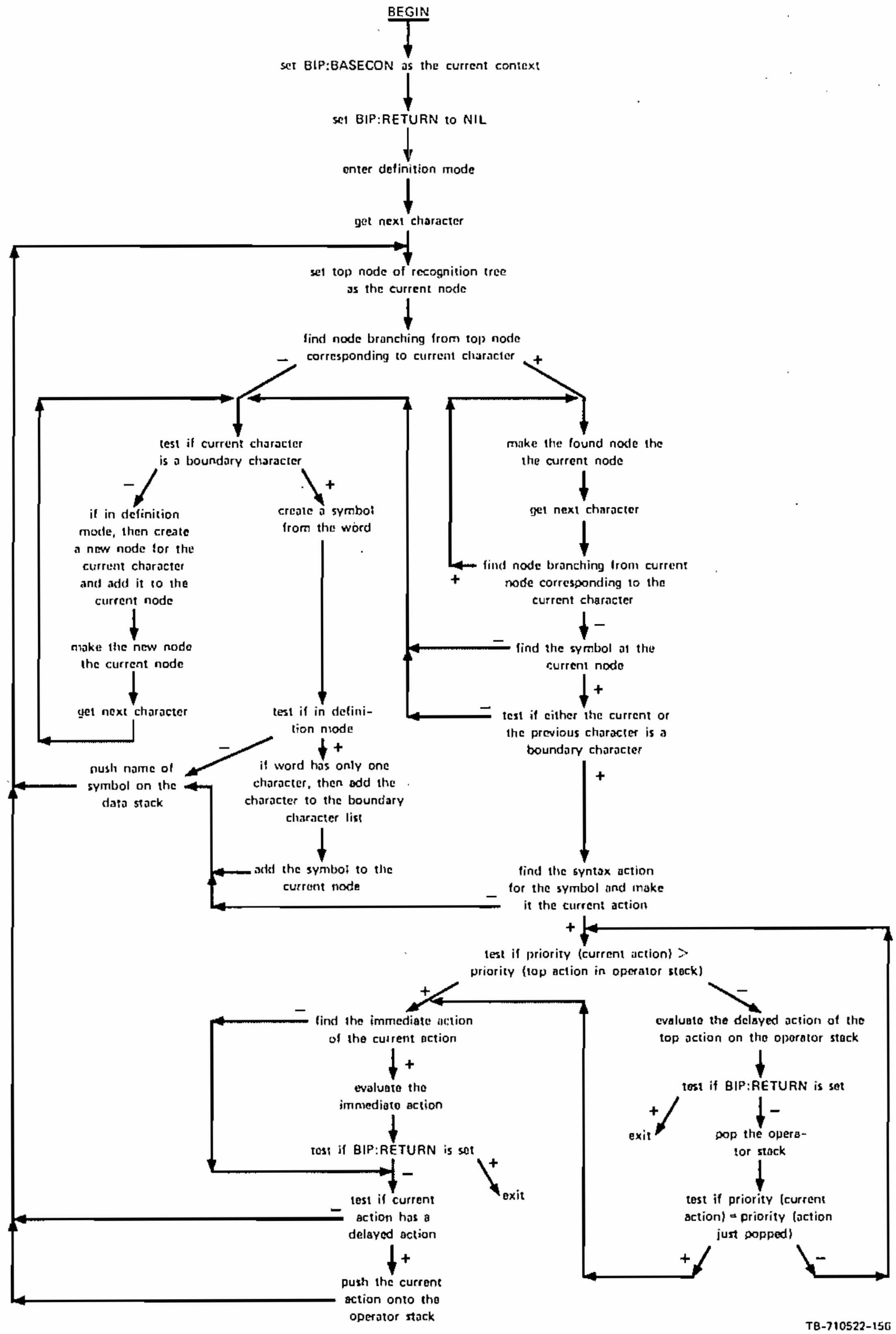

Figure 1 The BIP Translator