SCIENCE RESEARCH COUNCIL

RUTHERFORD AND APPLETON LABORATORIES
COMPUTING DIVISION

SOFTWARE ENGINEERING TECHNICAL PAPER 6

The Implementation of the Tree-Meta System on the ICL 1900

Issued by
J Malone
19 August 1980

Distribution: F R A Hopgood
R W Witty
D A Duce
W P Sharpe
C Prosser
S-E Technical Papers file

(Retyped by RWW Nov 2001)

1. Introduction

For each rule I the input definition a body of code is output labelled with the name of that rule. The first line of code is:-

ENTRY 0: %INIT; %CALL; DATA (@PROGM);

Where 'PROGM' is the name of the main syntax rule.

In a PLASYD program the return address for a procedure is stored in its link accumulator (which is any one of the registers X0 to X7), which appears in brackets after the procedure heading. So it is possible to jump to another routine after leaving a procedure by changing the value of the link accumulator.

Constant values may be stored at any point in a PLASYD program by writing DATA (<constant value>). Most routines have their arguments passed in this way, e.g. %PROC; DATA (<argument>) – the procedure %PROC will pick up its argument by accessing the item at its return address. Before returning, the return address is incremented to jump over the argument.

2. DATA STRUCTURES

2.1 String Table (SST(30000))

TYPE		LENGTH 4	
A	B	C	D
TYPE		LENGTH 3	
A	B	C	/
TYPE		LENGTH 6	
A	B	C	D
E	F	/	/
TYPE		LENGTH	
A	/	/	/
0		0	

TYPE	
0	STRING ON PROGRAM (DATA STMT)
1	ID
2	NUM
3	SR
4	HEX
5	OCT

Items placed in the string table are:-

(1) basic types e.g. .ID, .HEX
(2) strings preceded by . e.g. . 'STRING'
(3) strings in the actual syntax rule preceded by ^ e.g. ^'STRING'.

The procedure %STRE enters each string in SST. It first checks to see if the string is already in the table. %STRE returns a pointer to the string.

2.2 System Stack (MSTK(512))

Used to store return addresses and other temporary data such as pointers. %PUSHM and %POPM place items on the stack, %PUSHJ and %POPJ place the return address for syntax rules on the stack.

2.3 Stack (KSTK(128))

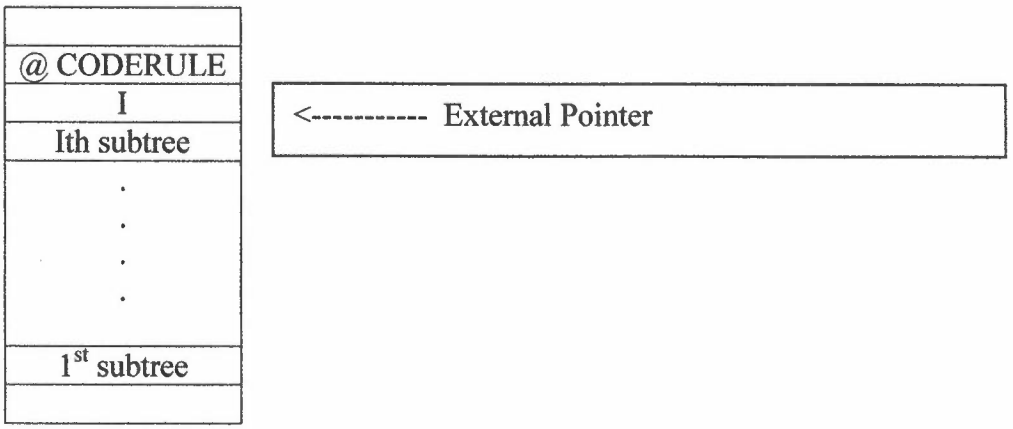
6	18
TYPE	ENTRY

Holds recognised items or pointers to items in the string table. These are used as leaves when building a subtree; the pointer to the subtree (which is held in the node table) will be placed on the stack so that it may be used to build a larger tree.

TYPE	
00	Character, letter or digit
02	Label
04	Pointer to string table (id, num, hex)
10	Pointer to string in string table
20	Pointer to node table

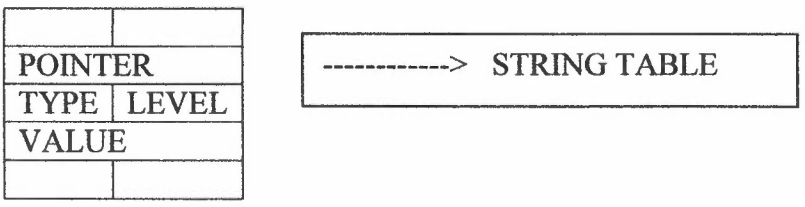
2.4 Node Table (NSTK(2048))

Contains tree node of form CODERULE[I]



When a tree node is generated, I items are taken off the stack and placed in the node table. These items may themselves be pointers to other nodes in the node table. The external pointer to this node, placed on the stack, points to the number of subtrees item.

2.5 Symbol Table (TSTK(1200))



Items are stored in increasing string pointer order. Type may be used to differentiate identifier types; level will refer to different block levels. When an entry is found using the procedure %LOOK, the entry with the largest level number is found. TREE-META translators may have up to 50 global integer variables. The nth one of these would be output as %A+n. These variables are stored with VALUE set to n.

2.6 Input Buffers

Each line of 80 characters is read into ILBUF then taken, a character at a time, into the RING buffer where it is used by the program.

(a) ILBUF (20)

Each world in ILBUF may contain 4 6-bit characters which are addressed by placing the world address in the bottom 22 bits of the addresss modifier and the character position (0,1,2,3) in the top 2.

The instruction INPTR=INPT + CHAR 1 will move INPTR to the next character position.

Using CH %ILBUF(INPTR) will access the character pointed at by INPTR.

The routine %INCHR reads in the next line when necessary and delivers the next character together with its type. The type is found by accessing a 64-character array, ILCONV, which contains the type value of every character.

(b) RING (256)

As characters are read into the ring buffer together with their type, spaces and comments are deleted and a switch set to show whether the input is a string.

The pointers used are:-

(1) %IWP which points to the current character being examined.
(2) %LASTC which shows the last character recognised.
(3) %DIFFC which hives the difference between the last character recognised and the last character in the buffer.
(4) %BACK which is set to 0 every time a rule allowing backtracking occurs and is recognised. It seems to be redundant, since it is not used to determine the input pointers when backtracking does occur.

%IWP and %DIFFC are incremented as each character is read into the buffer.

Before input is tested (i.e. for a string) %IWP is set to the last character recognised.

If recognised then %BACK is updated to include all characters to the end fo the buffer; %DEFFC is updated to be the difference between %IWP and the end of the buffer, then %LASTC becomes %IWP.

3.CODE GENERATED BY CONSTRUCTS IN DEFINITION

3.1 Syntax Rules

e.g. SRULE =

output:

```
SRULE:
 %PUSHJ
  .
  .
  .
code for rule
  .
  .
%POPJ
```

where %PUSHJ stacks the return address on MSTACK, and %POPJ returns control to this address.

3.2 Code Rules

e.g. CRULE =>

output:

```
DATA ("CRULE", 5);
CRULE;
%BEGN  or  %SBEGN
  .
  .
code for rule
  .
  .
%END   or  %SEND
```

%SBEGN and %SEND are used for simple rules of the form CRULE/=>. They use %AAA to store the return address.

%BEGN puts on MSTACK the return address, the pointer to the node in NSTACK which was the previous code rule and the label numbers used in hat rule, %END restores these values and returns to the stacked address.

3.3 Syntax Rule Structure

The syntax rules are mode up of a list of alternatives which consist of a series of subgoal tests (i.e. recognising strings or basic types on input) and directions for handling the tree (i.e. building and decoding nodes). Each subgoal test sets a flag, %MFLAG, so that if a goal is not recognised it is possible to branch on this flag to the end of the block of code for this alternative.

The block is structured in two ways, depending on whether backtracking is allowed:

(a) no backtracking

Failure to match the first subgoal of the block will result in branching to the end of the block. Subsequent failures will call %ERCHK which outputs an error message and stops.

e.g.

```
        .
        .
        .
        code for test 1
        %BF ; DATA (@%L1);
        .
        .
        code for test 2
%ERCHK, DATA (ERRNO)
        .
        .
%L1
```

(b) backtracking allowed (i.e. '->' appears at the beginning of alternative)

output:

```
        %SAV ;
        .
        .
        %L1:
        %RSTR;
```

where all failures to recognise subgoals cause a branch to %L1:

%SAV puts the following on MSTACK:

(i) %BACK. %BACK is then set to 0.
(ii) %LASTC
(iii) all entries in KSTACK
(iv) pointer to KSTACK (KPTR)
(v) pointer to NSTACK (NPTR)

If backtracking is not necessary (%MFLAG is tree) then %RSTR will remove these entries, using KPTR to tell how many KSTACK items were on MSTACK.

Otherwise %RSTR must restore NPTR (thus ignoring any new nodes which may have been built), KPTR and KSTACK. New input may have been read into the ring buffer, in which case the old value of %DIFFC will hav to be changed. %DIFFC (the difference between %LASTC and the end of the buffer) is given by subtracting the stacked value of %LASTC from the number of character in the buffer (%LASTC+%DIFFC). %LASTC and %BACK are then restored.

Output for alternative list:-

```
.
.
.
code for alternative 1
.
.
%L1:
%BT: DATA (@L2);
.
.
code for alternative 2
.
.
%L2:
```

Output for repetition (i.e. $(alternative list)) :-

```
%L1:
.
.
code for alternative list
.
.
%BT; DATA (@L1)
%SET;
```

so that the list is repeated as long as %MFLAG is et. %MFLA is rset by %SET on exiting from the loop.