

The PAL Programming Language

by

Martin Richards and others

`mr10@cl.cam.ac.uk`

`http://www.cl.cam.ac.uk/users/mr10/`

Computer Laboratory

University of Cambridge

Revision date: Wed Aug 21 07:06:05 AM BST 2024

Abstract

PAL was a programming language designed and implemented at MIT for use in a course to introduce first year students to how to write computer programs. [1]

This document is based on the source of the BCPL manual. Its main purpose is to provide the transition diagram for the syntax of PAL accepted by the PAL compiler and interpreter implemented by the program `BCPL/cintcode/com/pal.b`.

Contents

Preface	iii
1 The System Overview	1
2 The PAL Language	3
Bibliography	5
A PAL Syntax Diagrams	7

Preface

Chapter 1

The System Overview

Chapter 2

The PAL Language

Bibliography

- [1] M. Richards. *The Implementation of CPL-like programming languages*. Phd thesis, Cambridge University, 1966.

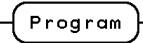




Appendix A

PAL Syntax Diagrams

This appendix gives the precise syntax of PAL as compiled by `pal.b`. It closely follows the syntax of PAL as implemented and maintained by Richard Mabee for the IBM 370, except curly brackets (`{ and }`) not parentheses are used to group declarations and command sequences. This greatly simplifies the syntax rules.

The syntax of programming languages is often specified using Backus Naur Form or BNF. Mathematicians like BNF notation because of its simplicity, power and interesting properties, while language designers like it because the rules just confirm their understanding of the language grammar they are designing. For users, understanding a grammar from its BNF specification is harder. There are typically a hundred or more of syntactic categories, many with rather artificial names, and a greater number of rules. Understanding the rules is hard because they mostly depend on each other. There is also sometimes a problem noticing whether a BNF grammar is ambiguous. Indeed it is not possible, in general, to write a program that can determine whether a BNF grammar is ambiguous, and it is also not always easy to write a parser that precisely agrees with the BNF specification. BNF was first used in the Algol 60 report and it suprisingly contained an ambiguity not noticed by the designers of that language.

Even though much research has been done in this area resulting in packages such as Lex and Yacc, I have decided to specify the grammar of PAL using a method based on transition diagrams. This method gives a precise specification of the parsing algorithm. The diagrams are easy to understand and have the advantage that the grammar is guaranteed to be unambiguous. It is also easy to check that the parser in a compiler conforms precisely with this specification. These diagrams can also be used as the basis of a program to find a minimum cost syntactic repairs, resulting in better error messages.

The PAL syntax is given using the diagrams shown in figures A.1, A.2, A.3, A.4 and A.5 for the syntactic categories **Prog**, **D**, **C**, **Bexp** and **En**. In the diagrams these categories are represented by the rounded boxes: , , , , and , respectively.

A rectangular box is called a test box and may contain a terminal symbol as in `-WHILE-` or `-<<-`, or a label representing a set of terminal symbols or some other condition. These test box labels are specified in the following table.

Label	Possible symbols or condition
name	A name not preceded by FLT
number	Integer or floating point constant
const	Integer or floating point constant, TRUE or FALSE
string	A string constant
mulop	* /
posop	+ -
addop	+ -
relop	= ~= < <= > >=
iscall	This is only satisfied if the most recent construct was a function
isname	This is only satisfied if the most recent construct was a name not enclosed in parentheses
eof	This is only satisfied if the program file is exhausted

A test box that specifies one or more terminal symbols can only be traversed if it matches the current symbol in the program. Some test boxes have side conditions such as `n<5` which must also be satisfied. When a box successfully matches a terminal symbol, input is advanced to the next symbol of the program.

Test and category boxes are connected by paths which may contain branch points where paths diverge, and join points where paths converge. Each diagram has an entry point and an exit point, and every path in it has an implied direction.

The diagrams specify an infinite extended flow graph obtained by starting with the category `-Program-` and repeatedly replacing category boxes by their definitions, substituting the parameter `n` where necessary. Every test box in the extended graph having a side condition will now compare two explicit integers and so is either equivalent to a test box without a side condition if the comparison is successful, otherwise the box will never succeed and can be eliminated. The extended graph thus only contains test boxes without side conditions.

The parsing algorithm searches through the extended flow graph trying to find a path containing a sequence of test boxes that match the terminal symbols of the program. Whenever a branch point is encountered, the left branch is tested first, only trying the right branch when all test boxes reachable from the left branch have failed. If a test box is successful and all boxes reachable from it fail, the program is syntactically incorrect. Parsing is thus done without any need to backtrack. A program is only syntactically correct if the exit point of the extended graph can be reached. The transition diagrams ensure that every loop in the extended graph contains at least one test block requiring a terminal

symbol to be read from the program. Parsing can therefore never be stuck in an infinite loop.

For compatibility with older versions of PAL some terminal symbols have synonyms as follow.

Symbol	Possible synonyms
NOT	~
= ~=	EQ NE
< <=	LS LE
> >=	GR GE
<< >>	LSHIFT RSHIFT
&	LOGAND LOGOR
> -*	

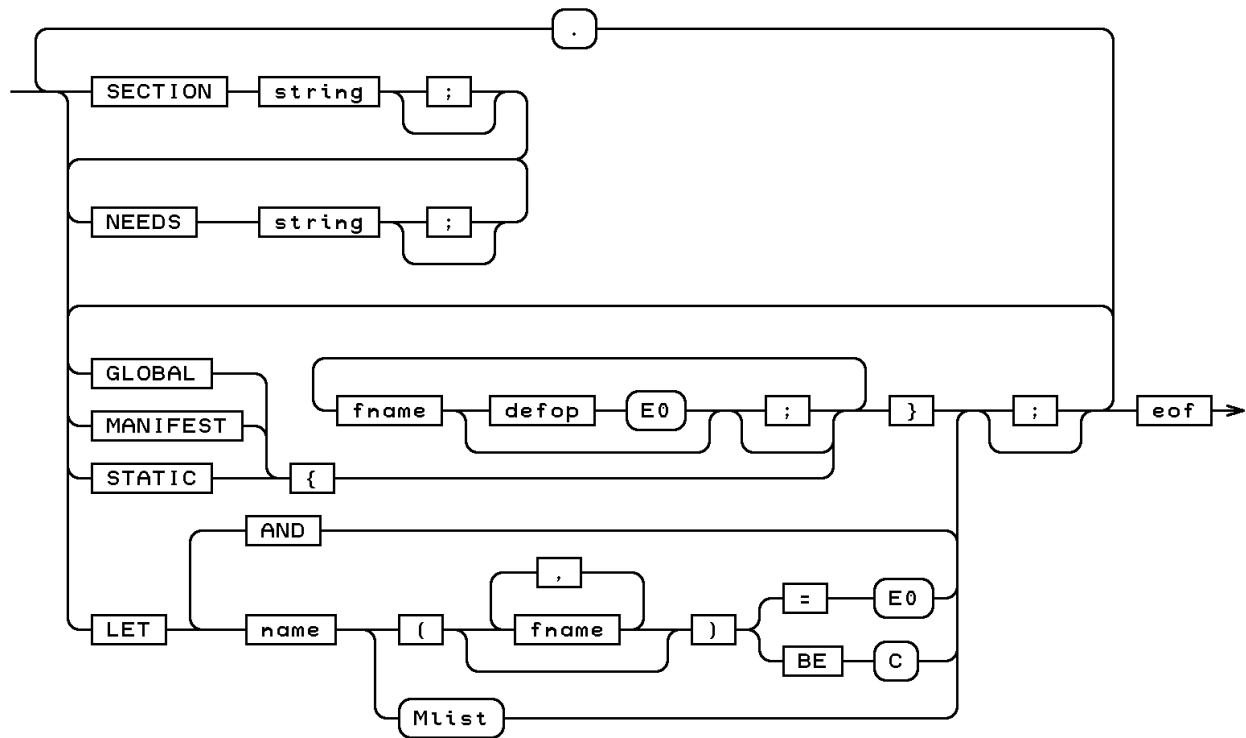
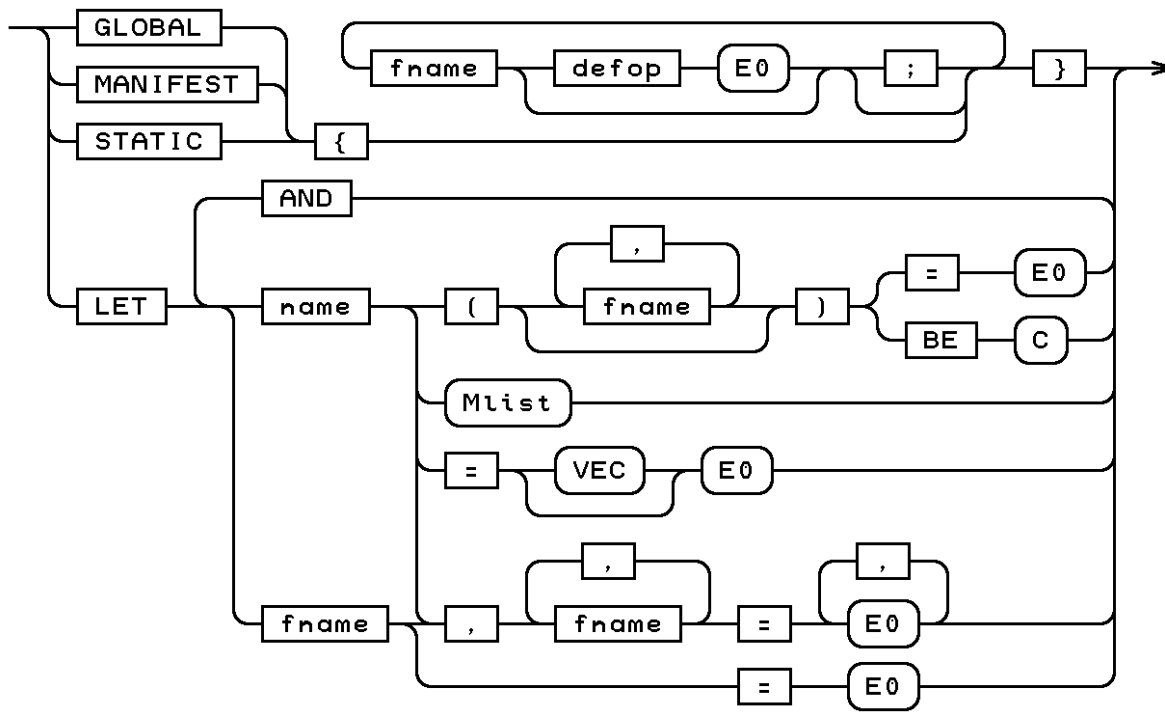
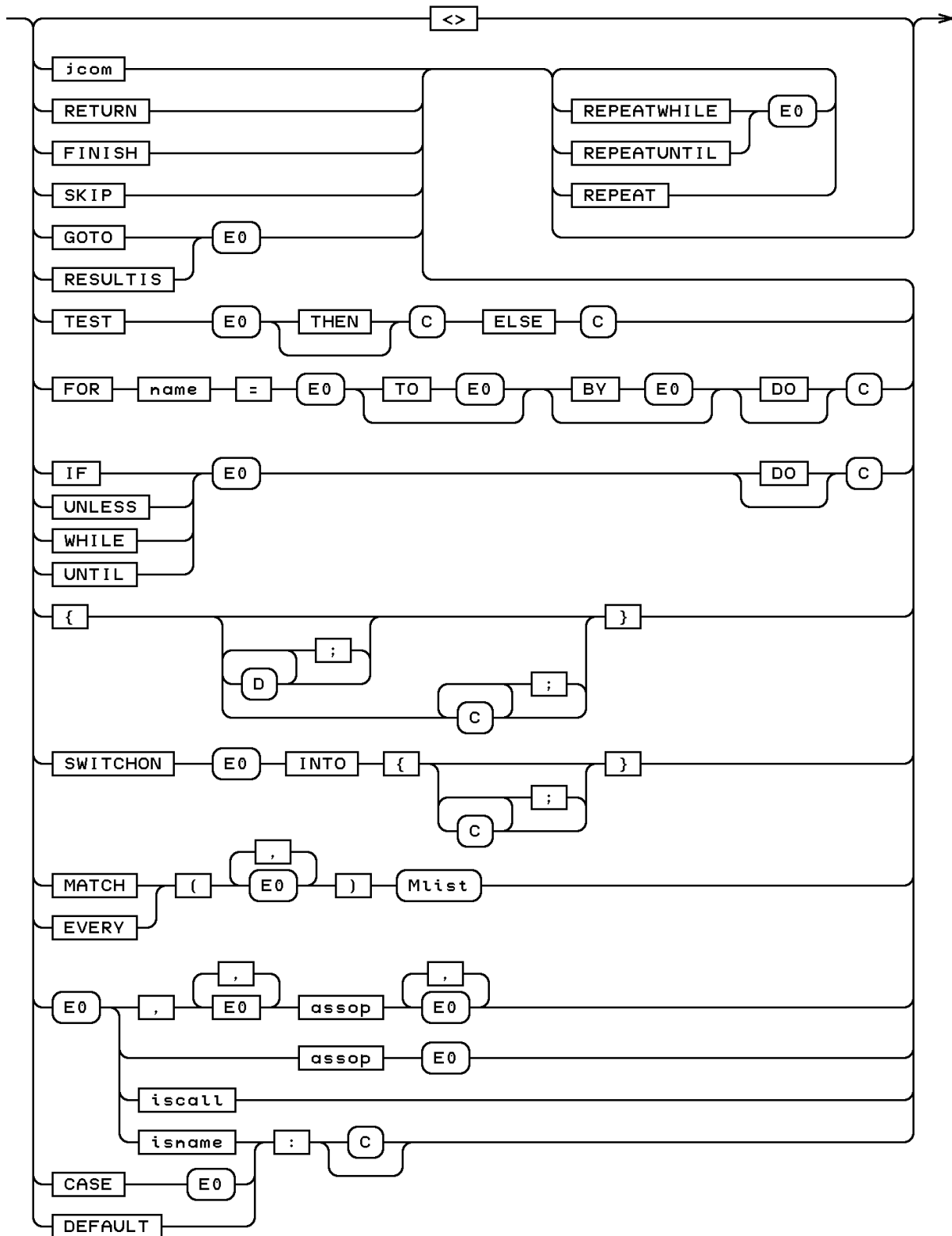
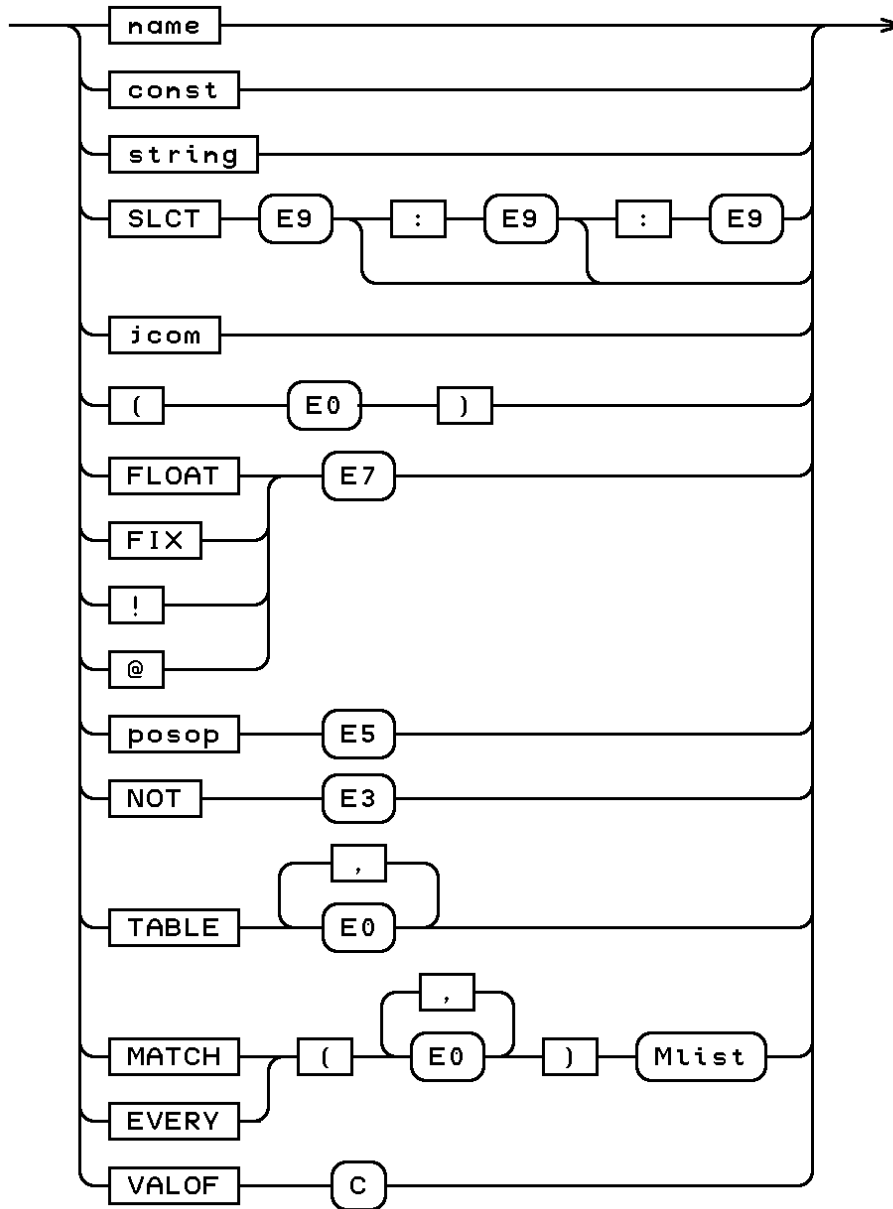
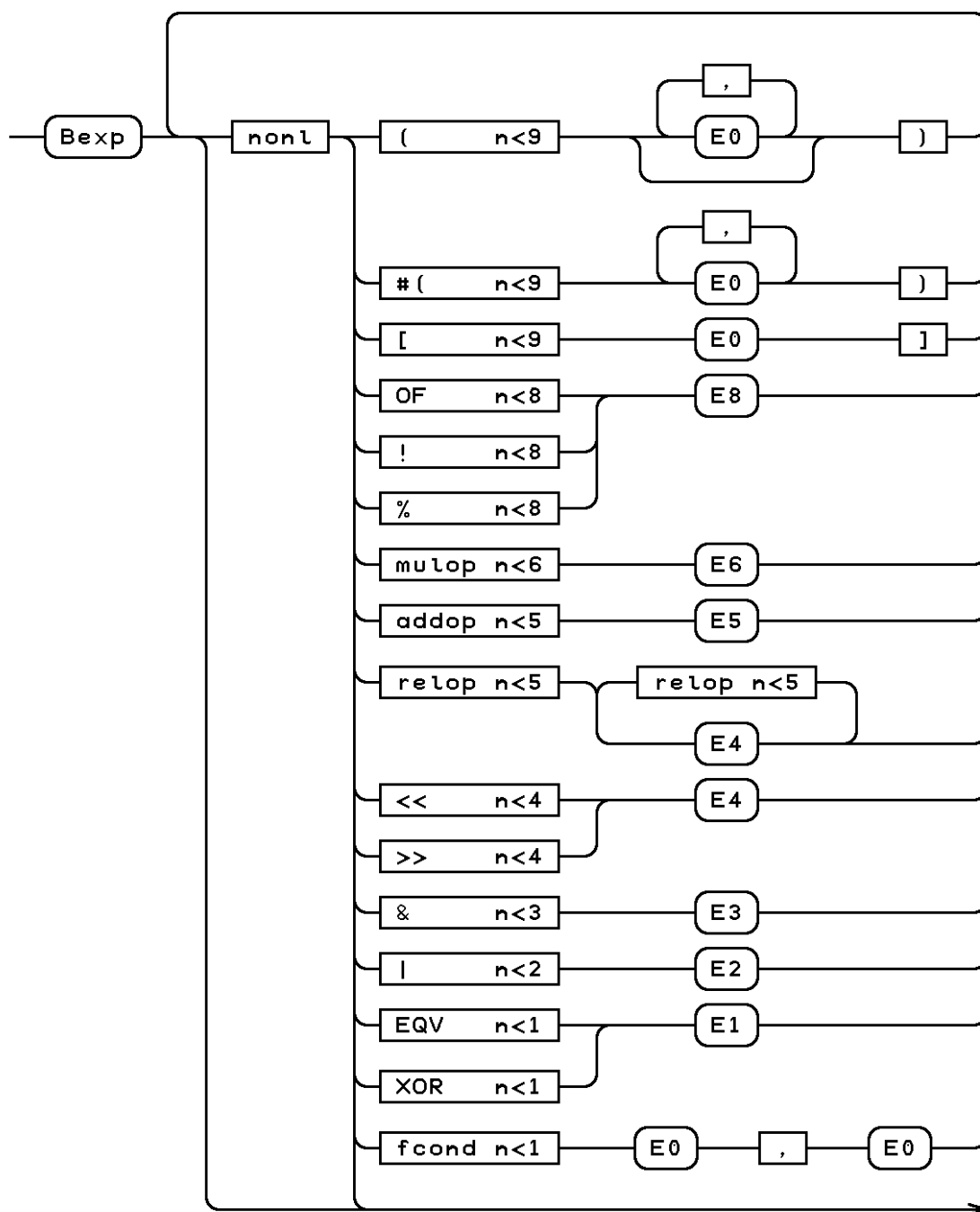


Figure A.1: The definition of `Program`

Figure A.2: The definition of [D]

Figure A.3: The definition of $\langle c \rangle$

Figure A.4: The definition of B_{exp}

Figure A.5: The definition of E_n

