/PEDIT - A Resident Structure Editor for PROLOG/

by

SANDRA LEE DUFFY

B.S. University of Illinois, 1977

A MASTER'S REPORT

submitted in partial fulfillment of the

requirements for the degree

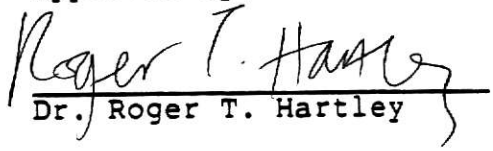
MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1985

Approved by:



Dr. Roger T. Hartley

PEDIT -
A Resident Structure Editor
for PROLOG

CONTENTS

1. Chapter One..... 1
   1.1 Introduction..... 1
   1.2 Literature Review..... 3

2. Chapter Two..... 12
   2.1 Requirements..... 12

3. Chapter Three..... 16
   3.1 Design..... 16

4. Chapter Four..... 27
   4.1 Implementation..... 27
   4.2 Testing..... 32

5. Chapter Five..... 33
   5.1 Conclusions..... 33
   5.2 Extensions..... 34

BIBLIOGRAPHY..... 37

Glossary..... 39

PEDIT Commands..... 40

USER'S MANUAL..... 42

Source Code..... 90

PEDIT -
A Resident Structure Editor
for PROLOG

LIST OF FIGURES

Figure 1. Program Flow..... 17

Figure 2. Structure Tree..... 18

Figure 3. Rule Tree..... 19

Figure 4. Fact Tree..... 19

Figure 5. PROLOG Structure..... 20

Figure 6. Rule Structure..... 20

Figure 7. Internal PEDIT Format..... 20

Figure 8. Goal Tree..... 22

Figure 9. Input Command Flow..... 29

1. Chapter One

PROLOG was created around 1970 [5] and as such is a relatively new programming language. This compares to 1962 for the creation of LISP, a language with similar applications. Due to its "youth", some of the utilities taken for granted in most programming environments are not available in the PROLOG programming environment. For example, an editor based within the PROLOG environment is non-existent to this author's knowledge. This paper describes PEDIT - a resident structure editor for PROLOG. Familiarity with PROLOG use and format by the reader is assumed. The implementation of PEDIT is intended to fill the editor gap in the PROLOG programming environment.

1.1 Introduction

Currently, a programmer debugging a program within the PROLOG environment has two options for making program changes. First, the programmer might exit the interpreter and edit the program file(s) using a text editor such as "vi" or "ed". Then, the interpreter must be reentered and the program file(s) reloaded. This iteration usually occurs many times before a program is correct. This method is both time-consuming and costly on any size program. On a large program, the costs could be prohibitive.

It should be noted that PROLOG commands can be created to edit program files without exiting the interpreter. This would give the impression of an interpreter-based editor; however, this method merely simulates the method mentioned above. It would not avoid the cost of reloading the edited files into the PROLOG environment.

A second method available to the PROLOG programmer is the manual deletion and addition of entire PROLOG clauses.* This could be accomplished with the use of the "retract" or "abolish" commands together with the "assert" commands. This method, however, would probably require the deletion and insertion of entire procedures due to the manner the "assert" commands operate.

If a single clause were "retract"ed and then "assert"ed, it would lose its position in the data base in relation to the other clauses for that procedure. Because PROLOG performs a "top-down" traversal of the data base, the logic for that procedure would in all probability be severely modified. Thus, entire procedures would need to be deleted and then inserted to ensure proper ordering of clauses. As can be seen, this method would only be practical if the entire

---

* NOTE: Appendix A contains a glossary of PROLOG terminology used in this paper.

procedure were faulty. If only a small error existed in a procedure, this method would be too cumbersome and time-consuming. Also, to an inexperienced typist, it could indeed resemble Purgatory.

Due to the reasons listed above, plus the enticement of enhancing an "infant" language, the task was undertaken to write a PROLOG-based editor. Since PROLOG is replacing LISP in some areas and thus is growing in use, it was agreed that this task would indeed fill a need of the future.

Once the task was undertaken, a decision was needed on what type of editor was appropriate. Since no PROLOG editor was known to exist, it was not possible to base the editor on other PROLOG editors. First, a general comparison of editor types was conducted. Second, since PROLOG and LISP are both used for Artificial Intelligence applications and as such are often used by the same personnel, thought was given to basing the PROLOG editor on existing LISP editors. Thus, a comparison of LISP editors was also conducted.

1.2 Literature Review

The merits and deficiencies of different types of editors have been debated since their invention. The viewpoints of various authors are presented and summarized here.