

Global Program Analysis in an Interactive Environment

by Larry Melvin Masinter

SSL-80-1 JANUARY 1980

Abstract: See next page

This report reproduces a dissertation submitted to the Department of Computer Science and the Committee on Graduate Studies of Stanford University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Key words and phrases: programming environments, cross reference, flow analysis, type inference, Lisp, program maintenance, natural language interface to data bases.

XEROX

PALO ALTO RESEARCH CENTER

3333 Coyote Hill Road / Palo Alto / California 94304

Abstract

This dissertation describes a programming tool, implemented in Lisp, called SCOPE. The basic idea behind SCOPE can be stated simply: SCOPE analyzes a user's programs, remembers what it sees, is able to answer questions based on the facts it remembers, and is able to incrementally update the data base when a piece of the program changes. A variety of program information is available about cross references, data flow and program organization. Facts about programs are stored in a data base; to answer a question, SCOPE retrieves and makes inferences based on information in the data base. SCOPE is *interactive* because it keeps track of which parts of the programs have changed during the course of an editing and debugging session, and is able to automatically and incrementally update its data base. Because SCOPE performs whatever re-analysis is necessary to answer the question when the question is asked, SCOPE maintains the illusion that the data base is always up to date—other than the additional wait time, it is as if SCOPE knew the answer all along.

SCOPE's foundation is a representation system in which properties of pieces of programs can be expressed. The objects of SCOPE's language are pieces of programs, and in particular, definitions of symbols—e.g., the definition of a procedure or a data structure. SCOPE does not model properties of individual statements or expressions in the program; SCOPE knows only individual facts about procedures, variables, data structures, and other pieces of a program which can be assigned as the definition of symbols. The facts are relations between the name of a definition and other symbols. For example, one of the relations that SCOPE keeps track of is **Call**; **Call[FN_1, FN_2]** holds if the definition whose name is FN_1 contains a call to a procedure named FN_2 .

SCOPE has two interfaces: one to the user and one to other programs. The user interface is an English-like command language which allows for a uniform command structure and convenient defaults; the most frequently used commands are the easiest to type. All of the power available within the command language is accessible through the program interface as well. The compiler and various other utilities use the program interface.

Preface

This dissertation is based on work the author did as part of the INTERLISP system [Teitelman, et al. 1978], and in particular, the MASTERSCOPE facility. MASTERSCOPE was designed and implemented entirely by the author. The basic idea for MASTERSCOPE was originally suggested by Warren Teitelman and a preliminary non-incremental version (called INTERSCOPE) was implemented by Phillip C. Jackson; a tree structure display program (called PRINTSTRUCTURE) had previously been implemented by Danny Bobrow. MASTERSCOPE was first completed in 1975, and has been in use by many INTERLISP users since then. The system described in this dissertation, called SCOPE, is a generalization and extension of MASTERSCOPE. While MASTERSCOPE was designed to be a robust tool for use by a large community, the emphasis in the design of SCOPE has been on improved functional capabilities; some of the efficiency and robustness has been sacrificed for its additional capabilities. There are currently no plans to make SCOPE generally available.

This work would not have been possible without the help of many people. I would like to thank in particular:

Warren Teitelman, for his early willingness to set me free on a problem, and for being the source of many of the ideas which profoundly influenced this work;

Terry Winograd, for his patient and careful readings of multiple drafts, and his support and encouragement;

Danny Bobrow and Bruce Buchanan, as well as Peter Deutsch, Cordell Green, Ron Kaplan, and Beau Sheil, for listening and reading;

Bob Taylor and the Xerox Palo Alto Research Center for financial support and incentives to finally be done; and

Carol Masinter, for editing, proofreading, and sharing with me for what has been a very long time.

Thank you.

