

Analysing FORTRAN Codes for Program Understanding and Reengineering

Warren O. Mason and Ronald B. Finkbine, Ph.D.

wmason@babbage.sosu.edu and finkbine@babbage.sosu.edu

Department of Computer Science
Southeastern Oklahoma State University
Durant, OK 74701
(405) 924-0121

Abstract

Legacy software represents a large investment for many organizations in the scientific community. The maintenance programmers, which keep these software systems operational are continually developing, modifying and updating code. In order to assist the maintenance programmer in understanding legacy software, it is desirable to have a software tool that can automate repetitive and computable tasks. Past research has often used the term “intelligent editor” to describe this function. This research project takes this direction further, it attempts to build an expert system that is able to understand segments of legacy software. The field of *Program Understanding* attempts to determine the function of a code segment with or without programmer intervention. This research analyses FORTRAN numerical analysis programs for common algorithm usage and other software characteristics. This paper discusses the problems identified by a cursory analysis of FORTRAN programs from the *Collected Algorithms of the ACM*.

Introduction

The purpose of this research is to develop a general-purpose algorithm recognition system, capable of recognizing any well-defined and well-written algorithm. This project uses models, (*plans* Wills, 1990) to recognize common forms (code segments) within existing software in an attempt to gain knowledge about a legacy software system. To understand legacy software, maintenance programmers generally review all existing documentation; however, many software systems will have little up-to-date documentation. This general lack of accurate external documentation results in the situation where the best documentation for a legacy system is the code itself (Biggerstaff, 1990). Program understanding generally occurs by matching the code segment in question against a large, defined set of models, or common algorithms, rather than attempting to deduce what a code segment performs from its specific data flow (Rugabur, 1990).

The ARS (Algorithm Recognition System) is an expert system which performs three functions: 1) analyzes input source code input by the user against a library of model numerical functions, 2) contains a library of models of numerical functions for comparison and detection, and 3) allows a user to input source code

that will become a model and then added to the existing library.

Background

Program understanding is a subfield of artificial intelligence and software engineering known by alternative names, such as software re-engineering and knowledge-based software engineering. The main thrust of research in program understanding is to discover general knowledge about a segment of source code. There are two possible uses for this type of software tool: 1) allow the developer of new software to compare the segment of code being tested to a known correct model within the expert system, and 2) allow the maintenance programmer an assistant for segmenting a large program under revision into known and unknown segments. It is hoped that the maintenance programmer will be able to make revisions, improvements, and enhancements to existing source code if an expert system is able to answer questions about the source code.

Targeted Problems

Legacy software, in general, exhibits a number of the following problems: 1) parameter identification, 2) identifying code segment that are replaceable by calls to commercial libraries (such as IMSL), 3) removing duplicate code to user library, 4) separation of

intertwined codes, and 5) combining disparate codes into single equations.

The first problem, *parameter identification*, is the most simple. It involves searching the source code for variables that are assigned values within assignment statements (no reads) one time. Any usage, thereafter, is only on the right-hand side of assignment statements and is reference to the variable, not a modification to the variable. Therefore, these types of variables, or constants, can be identified by the parameter statement which indicates their true usage.

The second problem, *plan recognition*, is comprised of identifying code segments that are replaceable by calls to commercial libraries (such as IMSL). This will involve detecting codes similar to those used within commercial libraries.

The third problem, *duplicate removal*, consists of detecting and removing duplicate code to the user's library. This allows the user to designate a section of code as common and to look through their remaining programs searching for codes that are copies of the target.

The fourth problem, *algorithm separation*, involves detection/separation of overlapping algorithms within the same section of code. In **Figure 1** it can be seen that there are two initializations of arrays occurring within the same do-loop. This is good for optimizing computer resources, but not for optimizing the programmers' time for understanding and maintaining a program.

```

DO 10 I = 1, N
    A(I) = 0
    B(I) = 0
10 CONTINUE

```

Figure 1: Intertwined Algorithms

The fifth problem, *algorithm aggregation*, involves combining disparate codes into single equations. As displayed in **Figure 2**, an equation 1) can be coded in multiple ways. Though the computations are equivalent, the recognition of them must take these variations into account.

```

1) distance = sqrt(side1*side1+side2*side2)
2) a = side1 * side1
3) b = side2 * side2
4) distance = sqrt(a+b)

```

Figure 2: Equation Variations

Detailed Example

The example subroutine in Figure is a portion of Algorithm 423 from the *Collected Algorithms From ACM*. The statements are identified for this discussion and a number of the statements are of interest. Statements 8 and 9 are the saving of an array position to a scalar variable. Statements 9, 10 and 11 are integrally related and constitute a swap, the exchange of values within two positions of the same array. Statements 12, 13 and 14 are a summation of a column within a matrix. Statements 18, 19 and 20 are a summation of a product of a scalar and a column within a matrix.

```

[ 1] SUBROUTINE SOLVE(N,NDIM,A,B,IP)
[ 2] REAL A(NDIM,NDIM),B(NDIM),T
[ 3] INTEGER IP(NDIM)
[ 4] IF (N.EQ.1) GOTO 9
[ 5] NM1=N-1
[ 6] DO 7 K = 1, NM1
[ 7]   KP1=K+1
[ 8]   M=IP(K)
[ 9]   T=B(M)
[10]   B(M) = B(K)
[11]   B(K) = T
[12]   DO 7 I = KP1, N
[13]     B(I)=B(I) + A(I,K)
[14] 7CONTINUE
[15] DO 8 KB = 1, NM1
[16]   KM1 = N - KB
[17]   B(K)=B(K) - A(K,K)
[18]   DO 8 I = 1, KH1
[19]     B(I)=B(I)+A(I,K) * T
[20] 8 CONTINUE
[21] 9 CONTINUE
[22] B(1)=B(1) / A(1,1)
[23] RETURN
[24] END

```

Figure 3: Algorithm 423

ARS Overview

The ARS system is formed of a number of modules as shown in Figure 1. The processing of FORTRAN source programs begin with traditional compiler

technology, lexical scanning and parsing. The input programs (whole programs, subroutines or functions) are transformed into an Internal Representation (IR) which resides within a Relational Database Management System (RDBMS). The use of a standard database allows for a correct and complete representation of data in much larger portions than traditional compilers. A 10,000 line program is going to produce an extremely large amount of data.

This project currently has a number of its major components under revision. The RDBMS is complete and stable. The FORTRAN scanner and parser are complete and working properly. Elementary recognition of models has been performed. The expert system shell used for the preliminary recognition is being replaced by a custom written rule engine. This is necessary since usage of the DOS-based CLIPS expert system was not conducive to repetitively

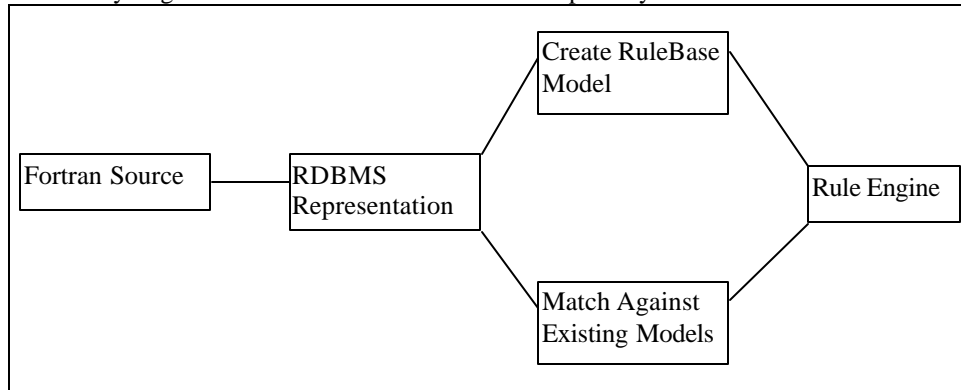


Figure 4:ARS System

From the IR one of two operations can be performed at user request. The most used operation will be plan recognition. This will analyze the user's source program, searching for models that have been pre-defined in the rule base system. The second operation available to the user is to be able to create a rule base for the specified input source segment. This allows the user to add a model to the models rule base and search source code for the designated model as well as the standard model library. This feature will allow the user to search for duplicate code within a single program or within an entire software system.

The rule system is divided into two subsystems: general-purpose and the user-specific library. The general-purpose library is a collection of models which are pre-inserted into the database. These include simple models such as an increment statement as well as models such as sorting algorithms.

Currently, this system is hosted on an IBM 80486 class machine with Windows 95 as an operating system. The ARS expert system itself is written in Delphi with a shareware RDBMS running in the background.

Current Status

executing for extremely small segments of code.

Acknowledgement

This project has been supported by the Faculty Research Fund at Southeastern Oklahoma State University.

Author Information

Mr. Mason is a undergraduate student in the B. S. program at SOSU. Dr. Finkbine is an Assistant Professor of Computer Science at Southeastern Oklahoma State University. He has a B.S. in Computer Science, 1985, and an M.S. in Computer Science, both from Wright State University. He has the Ph.D. in Computer Science from the New Mexico Institute of Mining and Technology.

References

Biggerstaff, Ted, *Design Recovery for Maintenance and Reuse*, in *IEEE Computer*, July, 1990.

Finkbine, Ronald B., *Recognition of High-Level Algorithms*, Ph.D. Dissertation, Department of Computer Science, New Mexico Institute of Mining and Technology, 1994.

Rugaber, Spencer, Stephen B. Ornburn, and Richard

LeBlanc, Jr., *Recognizing Design Decisions in Programs*, in *IEEE Computer*, July, 1990.

Wills, Linda, *Automated Program Recognition by Graph Parsing*, Ph.D. Dissertation, MIT Artificial Intelligence Laboratory, July, 1992.