Path-Oriented Program Analysis

This book presents a unique method for decomposing a computer program along its execution paths, for simplifying the subprograms so produced, and for recomposing a program from its subprograms. This method enables us to divide and conquer the complexity involved in understanding the computation performed by a program by decomposing it into a set of subprograms and then simplifying them to the furthest extent possible. The resulting simplified subprograms are generally more understandable than the original program as a whole. The method may also be used to simplify a piece of source code by following the path-oriented method of decomposition, simplification, and recomposition. The analysis may be carried out in such a way that the derivation of the analysis result constitutes a correctness proof. The method can be applied to any source code (or portion thereof) that prescribes the computation to be performed in terms of assignment statements, conditional statements, and loop constructs, regardless of the language or paradigm used.

J. C. Huang received a Ph.D. in electrical engineering from the University of Pennsylvania in 1969. He is a Professor Emeritus in the Department of Computer Science at the University of Houston, where he served as chair from 1992 to 1996.

His practical experience in computer software includes serving as the chief architect of a software validation and verification system developed for the U.S. Army's Ballistic Missile Defense Command, and as a senior consultant to the U.S. Naval Underwater Systems Center on submarine software problems.

Path-Oriented Program Analysis

J. C. Huang
University of Houston, Houston, Texas

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo, Delhi

Cambridge University Press
32 Avenue of the Americas, New York, NY 10013-2473, USA

www.cambridge.org
Information on this title: www.cambridge.org/9780521882866

First published 2008

Printed in the United States of America

A catalog record for this publication is available from the British Library.

Library of Congress Cataloging in Publication Data

Huang, J. C., 1935–
Path-oriented program analysis / J. C. Huang
p. cm.
Includes bibliographical references and index.
ISBN-978-0-521-88286-6 (hardback)
1. Computer software – Development. 2. Computer software –
Development – Computer programs. I. Title.
QA76.76.D47H83 2008
005.1 – dc22 2007026404

ISBN 978-0-521-88286-6 hardback

To my wife

Contents

Contents

Preface

Many years ago, I was given the responsibility of leading a large software project. The aspect of the project that worried me the most was the correctness of the programs produced. Whenever a part of the product became suspect, I could not put my mind to rest until the product was tested successfully with a well-chosen set of test cases and until I was able to understand the source code in question clearly and completely. It was not always easy to understand. That was when I started to search for ways to facilitate program understanding.

A program can be difficult to understand for many reasons. The difficulty may stem, for example, from the reader's unfamiliarity with the application area, from the obscurity of the algorithm implemented, or from the complex logic used in organizing the source code. Given the different reasons that difficulty may arise, a single comprehensive solution to this problem may never be found.

I realized, however, that the creator of a program must always decompose the task to be performed by the program to the extent that it can be prescribed in terms of the programming language used. If the

reader could see exactly how the task was decomposed, the difficulty of understanding the code would be eased because the reader could separately process each subprogram, which would be smaller in size and complexity than the program as a whole.

The problem is that the decomposition scheme deployed in any program may not be immediately obvious from the program text. This is so because programmers use code sharing to make source code compact and avoid unnecessary repetition. Code sharing, together with certain syntactic constraints imposed by programming languages, tends to obscure the decomposition scheme embodied in any program. Some analysis is required to recover this information.

Mathematically speaking, there are three basic ways to decompose a function. The first way is to divide the computation to be performed into a sequence of smaller steps. The second way is to compute a function with many arguments in terms of functions of fewer arguments. The third way is to partition the input domain into a number of subdomains and prescribe the computation to be performed for each subdomain separately. Methods already exist to recover and exploit information relevant to the first two decomposition schemes: They are known as the techniques of symbolic execution and program slicing, respectively. This book presents an analysis method that allows us to extract, analyze, and exploit information relevant to the third decomposition scheme.

Do not be intimidated by the formalisms found in the text. The theorems and corollaries are simply rules designed to manipulate programs. To be precise and concise, formulas in first-order predicate calculus are used to describe the rules. Only elementary knowledge of symbolic logic is needed to interpret those rules.

Typically, the method described in this book is to be used as follows. The program in question is test-executed with an input. If the program produces an incorrect result, it is a definite indication that the program is faulty, and appropriate action must be taken to locate and correct the fault. On the other hand, if the program produces a correct result, one can conclude with certainty only that the program is correct for that particular input. One can broaden the significance of the test result, however, by finding the execution path traversed during the test-execution and then applying the analysis method presented in this book to determine

(1) the conditions under which the same path will be traversed, and (2) the exact nature of the computation performed during execution. This information about execution paths in the program can then be integrated to obtain a better understanding of the program as a whole. This method is illustrated in Appendix A with example programs in C++.

This book contains enough information for the reader to apply the method manually. Manual application of this method, however, is inevitably tedious and error prone. To use the method in a production environment, the method must be mechanized. Software tool designers will find the formal basis presented in this work useful in creating a detailed design.

Being able to understand programs written by others is of practical importance. It is a skill that is fundamental to anyone who reuses software or who is responsible for software quality assurance and beneficial to anyone who designs programs, because it allows designers to learn from others. It is a skill that is not easy to acquire. I am not aware of any academic institution that offers a course on the subject. Typically, students learn to understand programs by studying small examples found in programming textbooks, and they may never be challenged, while in school, to understand a real-world program. Indeed, I have often heard it said – and not only by students – that if a program is difficult to understand, it must be badly written and thus should be rewritten or discarded. Program analysis is normally covered in a course on compiler construction. The problem is that what is needed to make a compiler compile is not necessarily the same as what is needed to make a programmer understand. We need methods to facilitate program understanding. I hope that publication of this book will motivate further study on the subject.

I would like to take this opportunity to thank William E. Howden for his inspiration; Raymond T. Yeh for giving me many professional opportunities that allowed this method to develop from conception, through various stages of experimentation, and finally to the present state of maturity; and John L. Bear and Marc Garbey for giving me the time needed to complete the writing of this book. I would also like to thank Heather Bergman for seeking me out and encouraging me to publish this work and Pooja Jain for her able editorial assistance in getting the book produced. Finally, my heartfelt thanks go to my daughter, Joyce, for her

active and affectionate interest in my writing, and for her invaluable help in the use of the English language, and to my wife, Shihwen, for her support, and for allowing me to neglect her while getting this work done.

J. C. Huang
Houston