Preface

Aims of the book. Our motivation for writing this book is twofold. One is that for quite some time we have lacked a textbook for our own courses on program analysis; instead we have been forced to base the courses on conference papers supplemented with the occasional journal paper or chapter from a text book out of print. The other is the growing feeling that the various subcommunities in the field often study similar problems without being sufficiently aware of insights or developments generally known in other subcommunities. The idea then emerged that perhaps a text book could be written that both would be useful for advanced courses on program analysis and that would help increase the awareness in the field about the many similarities between the various approaches.

There is an important analogy to complexity theory here. Consider researchers or students looking through the literature for a clever algorithm or heuristics to solve a problem on graphs. They might come across papers on clever algorithms or heuristics for solving problems on boolean formulae and might dismiss them on the grounds that graphs and boolean formulae surely are quite different things. Yet complexity theory tells us that this is quite a mistaken point of view. The notions of log-space reduction between problems and of NP-complete problems lead to realising that problems may appear unrelated at first sight and nonetheless be so closely related that a good algorithm or heuristics for one will give rise to a good algorithm or heuristics for the other. We believe it to be a sad fact that students of programming languages in general, and program analysis in particular, are much too often allowed to get away with making similarly naive decisions and arguments. Program analysis is still far from being able to precisely relate ingredients of different approaches to one another but we hope that this book will help to bring this about. In fact, we hope to shock quite a number of readers by convincing them that there are important similarities between the approaches of their own work and other approaches deemed to be of no relevance.

This book concentrates on what we believe to be the four main approaches to program analysis: Data Flow Analysis; Constraint Based Analysis; Abstract

Interpretation; and Type and Effect Systems. For each approach we aim at identifying and describing the important general principles, rather than presenting a cook-book of techniques and language constructs, with a view to how the principles scale up for more complex programming languages and analyses.

As a consequence we have deliberately decided that this book should not treat a number of interesting approaches, some of which are dear to the heart of the authors, in order to be able to cover the four main approaches to some depth; the approaches not covered include denotationally based program analysis, projection analysis, and logical formulations based on Stone dualities. For reasons of space, we have also had to omit material that would have had a natural place in the book; this includes a deeper treatment of set constraints, fast techniques for implementing type based analyses, single static assignment form, a broader treatment of pointer analyses, and the interplay between analysis and transformation and how to efficiently recompute analysis information invalidated by the transformation.

How to read the book. The book is relatively self-contained although the reader will benefit from previous exposure to discrete mathematics and compilers. The main chapters are generally rigorous in the early parts where the basic techniques are covered, but less so in the later parts where more advanced techniques are covered.

Chapter 1 is intended to be read quickly. The purpose of the chapter is to give an overview of the main approaches to program analysis, to stress the approximate nature of program analysis, and to make it clear that seemingly different approaches may yet have profound similarities. We recommend reading through all of Chapter 1 even though the reader might want to specialise in only parts of the book.

Chapter 2 introduces Data Flow Analysis. Sections 2.1 to 2.4 cover the basic techniques for intraprocedural analysis, including the Monotone Frameworks as generalisations of the Bit Vector Frameworks, and a worklist algorithm for computing the information efficiently. Section 2.2 covers more theoretical properties (semantic correctness) and can be omitted on a first reading. The presentation makes use of notions from lattice theory and for this we refer to Appendix A.1, A.2 and parts of A.3. — Section 2.5 is a more advanced section that gives an overview of interprocedural analysis including a treatment of call string based methods and methods based on assumption sets; since Section 2.5 is used as a stepping stone to Chapter 3 we recommend to read at least up to Subsection 2.5.2. Section 2.6 is an advanced section that illustrates how the relatively simple techniques introduced so far can be combined to develop a very complex shape analysis, but the material is not essential for the remainder of the book.

Chapter 3 covers Constraint Based Analysis. The treatment makes a clear distinction between determining the safety of an analysis result and how to compute the best safe result; it also stresses the need to analyse open systems. Sections 3.1, 3.3 and 3.4 cover the basic techniques; these include coinduction which is likely to be new to most readers and we refer to the treatment in Appendix B (building upon Tarski's theorem as covered in Appendix A.4). Section 3.2 covers more theoretical properties (semantic correctness and the existence of best solutions) and can be omitted on a first reading. — Sections 3.5 and 3.6 extend the development so as to link up with the treatment of Data Flow Analysis. Section 3.5 shows how to incorporate Monotone Frameworks (Section 2.3) and Section 3.6 shows how to add context in the manner of call strings and assumption sets (Section 2.5).

Chapter 4 covers Abstract Interpretation in a programming language independent fashion in order to stress that it can be integrated both with Data Flow Analysis and Constraint Based Analysis. Section 4.1 introduces some of the key considerations and is used to motivate some of the technical definitions in later sections. Section 4.2 deals with the use of widening and narrowing for approximating fixed points and Sections 4.3 deals with Galois connections; this order of presentation has been chosen to stress the fundamental nature played by widenings but the sections are largely independent of one another. — Sections 4.4 and 4.5 study how to build Galois connections in a systematic manner and how to use them for inducing approximate analyses; the material is not essential for the remainder of the book.

Chapter 5 covers Type and Effect Systems which is an approach to program analysis that is often viewed as having a quite different flavour from the approaches covered so far. Section 5.1 presents the basic approach (by linking back to the Constraint Based Analyses studied in Chapter 3) and suffices for getting an impression of the approach. Section 5.2 studies more theoretical properties (semantic correctness) and Section 5.3 studies algorithmic issues (soundness and completeness of a variation of algorithm $\mathcal{W}$) and these sections can be omitted on a first reading. — Sections 5.4 and 5.5 gradually introduce more and more advanced Type and Effect Systems.

Chapter 6 presents algorithms for Data Flow Analysis and Constraint Based Analysis. The treatment concentrates on general techniques for solving systems of inequations. We emphasise the fact that, to a large extent, the same set of techniques can be used for a number of different approaches to program analysis. Section 6.1 presents a general worklist algorithm, where the operations on the worklist constitute an abstract data type, and its correctness and complexity is established. Section 6.2 organises the worklist so that iteration takes place in reverse postorder and the Round Robin Algorithm is obtained as a special case. Section 6.3 then further identifies strong components and iterates through each strong component in reverse postorder before considering the next.

Appendices A and C review the concepts from partially ordered sets, graphs and regular expressions that are used throughout the book. Appendix B is more tutorial in nature since coinduction is likely to be a new concept for most readers.

To help the reader when navigating through the book we provide a table of *contents*, a *list of tables*, and a *list of figures* at the front of the book and an *index* of the main concepts and an *index of notation* at the end of the book. The index of notation is divided into three parts: first the mathematical symbols (with an indication of where they take their parameters), then the notation beginning with a greek letter, and finally the notation beginning with a letter in the latin alphabet (regardless of the font used). The book concludes with a *bibliography*.

Our notation is mostly standard and is explained when introduced. However, it may be helpful to point out that we will be using "iff" as an abbreviation for "if and only if", that we will be using $\cdots[\cdots \mapsto \cdots]$ to mean both syntactic substitution as well as update of an environment or store, and that we will be writing $\cdots \to_{\text{fin}} \cdots$ for the set of finitary functions: these are the partial functions with a finite domain. We also use λ-notation for functions when it improves the clarity of presentation: $\lambda x.\cdots x \cdots$ stands for the unary function f defined by $f(x) = \cdots x \cdots$.

Most proofs and some technical lemmas and facts are in small print in order to aid the reader in navigating through the book.

How to teach from the book. The book contains more material than can be covered in a one semester course. The pace naturally depends on the background of the students; we have taught the course at different paces to students in their fourth year as well as to Ph.D.-students with a variety of backgrounds. Below we summarise our experiences on how many lectures are needed for covering the various parts of the book; it supplements the guide-lines presented above for how to read the book.

Two or three lectures should suffice for covering all of Chapter 1 and possibly some of the simpler concepts from Appendix A.1 and A.2.

Sections 2.1, 2.3 and 2.4 are likely to be central to any course dealing with data flow analysis. Three to four lectures should suffice for covering Sections 2.1 to 2.4 but five lectures may be needed if the students lack the necessary background in operational semantics or lattice theory (Appendix A.1, A.2 and parts of A.3). — Sections 2.5 and 2.6 are more advanced. One or two lectures suffice for covering Section 2.5 but it is hard to pay justice to Section 2.6 in less than two lectures.

Four or five lectures should suffice for a complete treatment of both Chapter 3 and Appendix B; however, the concept of coinduction takes some time to get used to and should be explained more than once.

Four or five lectures should suffice for a complete treatment of Chapter 4 as well as Appendix A.4.

About four lectures should suffice for a complete treatment of Chapter 5.

Two lectures should suffice for Chapter 6 but three lectures may be needed if large parts of Appendix C need to be reviewed.

We have always covered the appendices as an integrated part of the other chapters; indeed, some of the foundations for partial orders are introduced gently in Chapter 1 and most of our students have had some prior exposure to partial orders, graphs and regular expressions.

The book contains numerous exercises and several mini projects which are small projects dealing with practical or theoretical aspects of the development. Many of the important links and analogies between the various chapters are studied in the exercises and mini projects. Some of the harder exercises are starred.

Acknowledgements. We should like to thank Reinhard Wilhelm for his long and lasting interest in this project and for his many encouraging and constructive remarks. We have also profited greatly from discussions with Alan Mycroft, Mooly Sagiv and Helmut Seidl about their perspective on selected parts of the manuscript. Many other colleagues have influenced the writing of the book and we should like to thank them all; in particular Alex Aiken, Torben Amtoft, Patrick Cousot, Laurie Hendren, Suresh Jagannathan, Florian Martin, Barbara Ryder, Bernhard Steffen. We are grateful to Schloss Dagstuhl for having hosted two key meetings: a one-week meeting among the authors in March of 1997 (during which we discovered the soothing nature of Vangelis' *1492*) and our advanced course in November of 1998. Warm thanks go to the many students attending the advanced course in Dagstuhl, and to the many students in Aarhus, London, Saarbrücken and Tel Aviv that have tested the book as it evolved ever so slowly. Finally, we should like to thank Alfred Hofmann at Springer for a very satisfactory contract and René Rydhof Hansen for his help in tuning the LaTeX commands.

Aarhus and London
August, 1999

Flemming Nielson
Hanne Riis Nielson
Chris Hankin

Preface to the second printing. In this second printing we have corrected all errors and shortcomings pointed out to us. We should like to thank Torben Amtoft, John Tang Boyland, Jurriaan Hage and Mirko Luedde as well as our students for their observations.

Official web page. Further information about the book is available at the web page `http://www.imm.dtu.dk/~riis/PPA/ppa.html`. Here we will

provide information about availability of the book, a list of misprints (initially empty), pointers to web based tools that can be used in conjunction with the book, and transparencies and other supplementary material. Instructors are encouraged to send us teaching material for inclusion on the web page.

Lyngby and London
October, 2004

Flemming Nielson
Hanne Riis Nielson
Chris Hankin

http://www.springer.com/978-3-540-65410-0

Principles of Program Analysis

Nielson, F.; Nielson, H.R.; Hankin, C.

1999, XXI, 452 p., Hardcover
ISBN: 978-3-540-65410-0