PREFACE

As a young person, I thought it quite reasonable that one should try to understand every detail of the technology that underpinned the artifacts of everyday life. It seemed entirely appropriate that as I drove along in my automobile I should appreciate the relative merits of overhead camshafts and the alternative design based on pushrods and rocker arms. Alas, as the catalog of artifacts multiplied I realized that it was simply not possible to understand *everything*. I must now cheerfully confess that I am slightly vague on the difference between the NTSC color system used by television sets in the US, and the PAL system used throughout most of the rest of the world. In spite of reality curbing my more extreme ambitions, I still want to understand the details of the technologies that are central to my life as a software producer, or which are intrinsically interesting anyhow.

At about the same time that I realized that one cannot understand *everything* I made an even more startling discovery. There are two kinds of people: There are those who want to understand their technologies, and those who frankly don't give a damn. This book is written for people who are of the first kind.

.NET stands at the convergence of three of the hottest technologies for the future of computing. The three are: Component technology, web-based service provision, and language implementation based on abstract machines. Component technology will finally deliver on the promises of object-oriented programming. Web based service provision using industry standards such as XML and SOAP will power the new world of pervasive and mobile computing. Finally, program realization by means of abstract machines allows unprecedented levels of program portability as well as enabling verifier-based guarantees of runtime behavior. Add to this the broad industry support for standardization of .NET, and it becomes clear that .NET is set to dominate the immediate future of computing on the desktop and beyond.

The technology that underpins .NET, and enables all of the functionality that other books in this series will describe, is the *Common Language Runtime* (CLR). The CLR is the engine that powers the machine, the *Bernoulli theorem* that explains why the whole thing flies. For those who wish to understand how .NET works, this is the place to start.

In the final months of 1999 a small group of programming language implementation teams from around the world were invited to implement programming languages on what was to become the .NET runtime. The project produced a number of alternative languages to the ones that Microsoft themselves were developing, and showcased the multi-language capability of the CLR. Paul Roe and I produced a compiler for a Pascal language variant. I learned several things from the experience. I learned that the CLR is compiler-writer friendly, and that a small team can implement a real language for this target in a short time. As well as the learning, the project turned out to be great fun and I developed an enthusiasm for the CLR that I hope will prove to be infectious.

Since the public announcement of .NET and the public availability of the pre-release software, I have been involved in some experimentation with the use of the CLR in

education. Wayne Kelly has pioneered the use of the CLR as the target machine for an introductory compiler course at QUT, with encouraging results. For those colleges that advocate a practical approach to the compiler subject the CLR has very much to recommend it.

Since the technology described here is so new, it was difficult to find reviewers who were able to provide feedback during the development of this book. I owe a special thanks to the following .NET pioneers: Simeon Cran, Bertrand Meyer, Erik Meier, Christine Mingins, Paul Roe and Clemens Szyperski. As usual, any remaining errors are entirely my responsibility.

The compiler that forms the running example for this book grew out of a project jointly sponsored by the Distributed Systems Technology Center (DSTC), and the Programming Languages and Systems Research Center (PLAS) at QUT. Much of the underlying design for the compiler was developed in an earlier project supported by the Australian Research Council. Diane Corney and Sui-Yuen Chan, both doctoral alumni of PLAS, wrote some of the associated tools, and participated in the design reviews of the compiler.

Writing a book to a tight deadline plays havoc with ones social life. My family has been tolerant of the occasions when I have been absent, and other times when my body was present but my mind was far away. I owe a special thanks to my partner Noeleen Atwell for her love and support throughout the whole process.

John Gough
Brisbane 2001.