

Computer Science Tripos Project Proposal

Decompilation of .NET bytecode

Stephen Horne, Trinity Hall College

Originator: Jeremy Singer

23 October 2003

Special Resources Required

The use of my own PC (3GHz Pentium 4, 1Gb RAM and 192Gb Disk).

Project Supervisor: Eben Upton

Director of Studies: Dr S. W. Moore

Project Overseers: Dr J. G. Daugman & Dr J. M. Bacon

Introduction

Decompilation is the process of converting machine-executable code into a high-level language. There are several reasons why one might want to do this, such as checking software for malicious code, translating between different source languages, and recovering the lost source code of a legacy application. My intention is to create a decompiler for the Microsoft .NET platform. This platform is based on a similar idea to Java, in that code is compiled to an intermediate form before being interpreted on a stack-based virtual machine.

Such a program will be capable of taking a .NET executable file or dynamic link library, extracting the relevant Common Intermediate Language (CIL) intermediate code, and transforming this to an appropriate high-level language. Initially the high-level destination language will be C#, an object-oriented language created purely for .NET, but the software should be designed with the ability to add additional target languages easily. Ideally any other information necessary for recompilation should be extracted, such as resources contained within the executable.

The software itself will be divided into clearly separated and independent modules, each performing different stages of the decompilation. This will facilitate the expansion of the software in the future, in allowing new source and target languages to be added easily.

Work that has to be done

I will be basing my decompiler on Cristina Cifuentes's thesis "Reverse Compilation Techniques". However, her work describes the process for a register-based machine and so it is likely I will need to adapt much of this.

The first phase of the software will be responsible for loading in the CIL bytecode from either a .NET executable or dynamic link library. This bytecode will be syntactically analysed, identifying specific commands and data. Commands are analysed for their semantic meaning, identifying basic blocks within the code. Intermediate code that is not specific to the code being decompiled can be generated along with a control-flow graph, and these passed to the next phase of the software.

The second phase takes as its input the intermediate code and control flow graph generated by the first phase. Data-flow analysis will be performed on the low-level intermediate code, identifying complex expressions spread across several individual instructions and representing these using a high-level intermediate code. Control-flow analysis is performed on the control-flow graph generated in the first phase in order to identify high-level control structures such as decision branches and loops, giving a structured control-flow graph to pass into the final phase.

The third and final phase of the software reads in the high-level intermediate code from the previous phase, and uses this in conjunction with the structured control-flow graph to generate the target code. Since the second phase is source and target language independent, this phase should be as modular as the first. This means that different languages can be generated by using the appropriate final phase module; this is particularly useful for the .NET architecture since it is likely a user would want to be able to choose between the many .NET-enabled languages available.

Difficulties to Overcome

There are a couple of difficulties that I expect to encounter in the course of this project. At this stage I am unaware how much optimisation the C# compiler performs, and this may remove a lot of the useful control structure information. Also, initial research suggests CIL binaries do not contain type information and hence this will have to be inferred from the code.

Starting Point

As I begin this project I have very little knowledge of the Microsoft .NET framework or the CIL intermediate language. However, I have come across the Java Virtual Machine before this and based on my limited observation so far I believe the two to be similar in basic concept.

The project should be as modular as possible to facilitate development and its future expansion, and this suggests an object-oriented solution. Whilst Java would be a suitable language for structuring a piece of software such as this and is a language I know well, I have chosen to develop the decompiler in C# instead. This means that I will learn C# as the project progresses, and as I get to the stage of writing out decompiled C# I will hopefully have a good understanding of the language. My initial research suggests C# is very similar to Java and therefore should be straightforward to learn. Another advantage of writing the decompiler in a .NET-enabled language is that at the end I will have a large sample application to decompile (the decompiler itself).

Deliverables

For my project to be considered a success it should be able to produce recompilable C# from a basic .NET executable. The executable should represent a program containing multiple classes, making use of variables of different types as well as various methods, and should make use of simple branching and looping constructs. The recreated source should be recognisably similar to the original as well as functionally equivalent, and easily comprehended by someone who knows C#. Ultimately I hope that the decompiler will be capable of transforming itself into comprehensible C# code.

Beyond these basic deliverables there are other features that I would like to see implemented, such as the ability to decompile into .NET languages other than C# such as Visual Basic, as well as a good graphical interface to the software. These are secondary however to the task of getting a basic decompiler up and running.

Resources

I intend to use my own PC for this project, since it is reasonably powerful and has the Microsoft .NET SDK installed. I will be using CVS to manage the code, and keeping the repository backed up through the use of CD-R's and the Pelican system.

Plan of work

The following list gives a timetable of the various stages of the project that will need to be completed. The project is divided into 10 two-week work blocks, with a break for Christmas.

Michaelmas

- Block 1: 25/10/03 - 07/11/03
Learn C#, understand the general .NET framework, and learn the CIL intermediate language.
- Block 2: 08/11/03 - 21/11/03
Research various decompilation methods and create a detailed software plan.
- Block 3: 22/11/03 - 05/12/03
Implement the software 'front-end'; this involves reading in a compiled .NET program and transforming it to an appropriate internal representation.
- Block 4: 06/12/03 - 19/12/03
Implement the analysis phase; this will involve control and data flow analysis.

Lent

- Block 5: 10/01/04 - 23/01/04
Implement the software 'back-end'; this involves writing out the target code.
Write progress report and prepare presentation: deadline 30/01/04.
- Block 6: 24/01/04 - 06/02/04
Test core routines and identify any special cases.
- Block 7: 07/02/04 - 20/02/04
Polish off final details of the code.
If time, create a GUI for interaction with the software.
- Block 8: 21/02/04 - 05/03/04
Write dissertation up to and including Preparation section.
- Block 9: 06/03/04 - 19/03/04
Write Implementation and Evaluation sections of dissertation.
- Block 10: 20/03/04 - 02/04/04
Write Conclusions and Appendices sections of dissertation.
Have dissertation ready for handing in: deadline 14/05/04.