School of Computing final year project

Robin Eklind

PJE40

Project Initiation Document

Compositional Decompilation

Project Initiation Document

1. Basic details

Student name:	Robin Eklind
Draft project title:	Compositional Decompilation
Course:	ERASMUS Exchange Programme in the Faculty of Technology
Client organisation:	N/A
Client contact name:	N/A
Project supervisor:	Janka Chlebíková

2. Outline of the project environment and problem to be solved

Decompilation enables source code reconstruction of binary applications, and was originally referred to as reverse compilation. Both software engineers and security researchers may benefit from decompilation, as it facilitates analysis, modification, and reconstruction of object code.

The applications of decompilation are versatile. It is used to:

- analyze malware
- recover source code
- migrate software from legacy platforms
- optimize existing binary applications
- discover and mitigate bugs and security vulnerabilities
- verify compiler output with regards to correctness
- analyze algorithms
- improve interoperability with other software
- add new features to existing software

3. Project aim and objectives

The aim of this project is to facilitate decompilation workflows using composition of language-agnostic decompilation passes; specifically the reconstruction of high-level control structures and, as a future ambition, expressions.

In order to achieve this aim, the author will:

1. Review traditional decompilation techniques, including control flow analysis and data flow analysis.
2. Critically evaluate a set of Intermediate Representations (IR), which describes low-, medium- and high-level language semantics, to identify one or more suitable for the decompilation pipeline.
3. Analyze the formal grammar (language specification) of the IR to verify that it is unambiguous. If the grammar is ambiguous or if no formal grammar exists, produce a formal grammar. This objective is critical for language-independence, as the IR works as a bridge between different programming languages.

4. Determine if any existing library for the IR satisfies our requirements; and if not develop one. The requirements would include a suitable in-memory representation, and support for on-disk file storage and arbitrary manipulations (inject, delete, etc) of the IR.
5. Design and develop components which identify the structural patterns of high-level control structures using control flow analysis of the IR.
6. Develop tools which perform one or more decompilation passes on a given IR. The tools will be reusable by other programming language environments as their input and output is specified by a formally defined IR.
7. As a future ambition, design and develop components which perform expression propagation using data flow analysis of the IR.

4. Project deliverables

Documents to be produced:

- Project report.
- Formal grammar for the LLVM IR language; refer to objective 3.

System artefacts to be developed:

- Library for interacting with LLVM IR; refer to objective 4.
- Library for reconstructing high-level control structures from LLVM IR; refer to objective 5.
- Tool for performing one or more decompilation passes on a given IR; refer to objective 6.

5. Project constraints

The project may be limited by time constraints, lack of previous experience with decompilation, insufficient documentation of third party software, and ambiguous language specifications.

6. Project approach

Aim to become proficient in the key concepts and algorithms of decompilation before working on the design of the project artefacts. After reaching proficiency design a decompilation system composed of individual components and based on the principle of separation of concerns. The system must be language-agnostic so that decompilation passes can be reused from other programming language environments. Attempt to find flaws in the design by stress testing it through proof of concept implementations. Engage in discussions with the open source community during the design process of any library intended for third party use; specifically the LLVM IR libraries. Once the design is deemed mature, begin an iterative process of implementation and testing. Don't be afraid to throw away code and trying new ideas, that's what revision control systems are for! Use the Clang compiler to produce test cases, as it is capable of emitting LLVM IR from C source code. The goal will be to reconstruct the high-level control flow structures (such as for-loops, if-else statements, etc) of the original C code from the LLVM IR.

The LLVM IR libraries are developed as reusable components for compilers, decompilers, and other semantic analysis tools. They aim to support generic semantic analysis applications, while satisfying the explicit requirements[1] of the third party llgo compiler. The libraries will support all three forms of the LLVM IR format; which includes an in-memory Intermediate Representation, a human readable assembly language representation, and a binary bitcode file format.

All work will be made available[2][3] on GitHub from day one, and the source code will be released into the public domain to encourage open source adaptation. The project plan will be organized using the

[1] https://github.com/go-llvm/llvm/issues/40
[2] https://github.com/mewpaper/decompilation
[3] https://github.com/mewlang/llvm

GitHub issue tracker, where each issue corresponds to a task. Milestones, containing a set of issues, will track the progress of the project and enforce deadlines.

No single methodology will be used for this project, but rather a combination of techniques which have been proven to work well in practice. Continuous Integration will be used throughout the development process to monitor the build status, test cases, and code coverage of each component. The API stability of each component will be governed using Semantic Versioning (vMAJOR.MINOR.PATCH). Prior to version 1.0 the components will be written using either throw-away prototyping or iterative development and the API may change drastically. The implementation correctness will be verified using Test Driven Design where feasible, with test cases written prior to the actual implementation of the API.

7. Facilities and resources

The project requires at least one computer with access to the Internet and with the following open source software:

- A compiler for the Go programming language.
- Tools from the LLVM compiler infrastructure.
- LaTeX.

8. Log of risks

Insufficient time:
The main risk of this project is without doubt the lack of time, caused by a disproportionate allocation of time either between leisure activities and the project, or between tasks within the project itself. Proper time management is vital to mitigate this risk, and will be an ongoing effort throughout the project.

To facilitate time management each task will be tracked using the GitHub issue tracker, and larger tasks will be divided into suitably sized sub-tasks. The smaller sub-tasks help maintain focus and make it easier to establish reasonable deadlines. Repeatedly postponed or ignored sub-task deadlines would indicate an actual risk.

Implementation difficulties:
Difficulties with advanced algorithms and concepts may limit the progress of the project. To mitigate this risk implementation details will be considered during the literature review, and additional time will be allocated to the understanding of initially unintuitive algorithms. Incomprehensible key algorithms and concepts towards the end of the literature review would indicate a severe risk to the project.

Data loss:
To mitigate the risk of data loss all work will be stored both locally and remotely (GitHub), and synchronized regularly. A long time interval between synchronizations would indicate an actual risk.

9. Starting point for research

The following theses, papers, and online references will be included in the literature review:

- C. Cifuentes, *Reverse Compilation Techniques*. PhD thesis, Queensland University of Technology, 1994.
- A. Mycroft, *Type-Based Decompilation*, in 8th European Symposium on Programming, ESOP'99, pp. 208–223, Springer-Verlag, 1999.
- M. J. Van Emmerik, *Static Single Assignment for Decompilation*. PhD thesis, The University of Queensland, 2007.
- S. Moll, *Decompilation of LLVM IR*. BSc thesis, Saarland University, 2011.

- The LLVM Language Reference Manual.

10. Breakdown of tasks

A list of specific tasks is actively maintained in the project plan, which is hosted online.

11. Project plan

The project plan will be managed using the GitHub issue tracker, and actively maintained throughout the project. Each task (research topic, unresolved problem, software feature, etc) will be allocated a dedicated issue tracking its progress. The issues include relevant discussions of identified risks, potential problems and proposed solutions. The project plan provides an overview of the active issues, using milestones to group issues and assign deadlines. As the project plan is in an ever evolving state, an online version has been made available at:
https://github.com/mewpaper/decompilation/issues/1

12. Legal, ethical, professional, social issues

There are no known legal, ethical, professional, or social issues which may impose constraints on the project. No research ethics approval will be required for the project, as no experiments or studies will be conducted on human subjects. Please refer to the ethical examination checklist for further comments.

Signatures

	Signature:	Date:
Student		
Client	N/A	N/A
Project supervisor		