Bidirectional Compiler for Software Evolution

Meng Wang

1 Research Aim and Objectives

We aim to improve the process of software evolution by reducing the disruption caused by updating third-party software. It is common for downstream developers to customise and optimise third-party software to suit their intended use. But since the source code of the third-party software is usually unavailable to downstream developers, such customisation are often difficult and labour intensive. Moreover, there is a software evolution problem, as when the upstream software vendor releases an update, the downstream customisations to the existing version risk being overwritten. This disruption in software update is costly to users, and delays potentially safety- and security-critical updates being performed.

In this project, we address this problem by applying bidirectional-transformation techniques to compiler construction. By making compilers bidirectional, one will be able to automatically generate human-readable source code from the low-level executable. The backward generated source code will serve the following due objectives. 1. Being more susceptible to both human and automated reasoning, it will simplify the software customisation process, reducing manual effort and improve reliability. 2. Being consistent to the customised executable, it keeps the customisation, which will not be overwritten by incoming updates to the executable. The bidirectional-transformation process will then be able to “synchronise” the customised source code and the updated executable to automatically merge the customisation with the incoming update, minimising the need for manual intervention.

2 Research Proposal

Compilation is essential to software development, which transforms human-readable source programs into low-level code for machine execution. In today’s software industry, the vast majority of software is shipped with compiled code only, making the understanding and verification of them difficult. De-compilation is a solution to this problem. It can be seen as a kind of inverse of compilation, taking low-level code as input to produce corresponding high-level programs for analysis and verification, and in some cases customisation and optimisation to suit the user’s environment.

The fact that downstream users customise compiled code creates a software evolution problem. Every time when the third-party software vendor releases an update, typically in the form of a new version of the compiled code, the (painstakingly crafted) user-customisation will be overwritten. The threat of such disruption partly explains why many system admins are reluctant to perform updates, clinging on out-of-date software with safety issues and security vulnerabilities. Existing de-compilation is of no help here: despite its ability to construct high-level programs which facilitates the initial customisation, each de-compilation is done in isolation and therefore does not handle evolution.

In this project, we will introduce a new model of de-compilation, in the form of bidirectional transformations (BX), which originated as a solution to the database view-update-problem, and is applied to the broader area of data synchronisation. Unlike traditional de-compilation, BX is change based. It concerns the synchronisation of source data (say source program), and target data (say compiled code) that is generated (say compiled) from the source data. When the target data is changed by an external process (say user customisation), the backward transformation of the BX will create new source data with the changes incorporated. A correctness criterion (known as round-tripping) is that the new source data must be able to re-generate the changed target data. In other words, the change to the target data is “saved” in the source, which will be the basis for handling subsequent changes.

This is an ambitious undertaking, as no BX of this complexity has ever been constructed. But it is at the same time a natural next step from the foundation built by the applicant. The applicant is a leading expert on BX and pioneered the idea of using BXs to connect the stages of compiler by viewing the different representations of the program under compilation as data and the compilation processes as bidirectional. As the applicant already has experience with bidirectionalizing compiler front-end, including preprocessing, desugaring, and parsing, the focus will be on the middle-end which mostly concerns optimisations. We will exploit the non-compulsory nature of optimisations to devise a relaxed version of round-tripping (i.e., two compiled programs are considered “equal” if they differ only in performance), which we anticipate to be necessary for handling complex program transformations. We will also explore the different back-end types. For example, the more abstract bytecode is likely to be a more suitable target than native machine code.

Success will be measured by the construction of a BX for a simple but realistic compiler, and the development of new BX techniques for achieving it. This will represent significant advance of the theory and practice in BX and de-compilation research. In the mid-term, this new approach will likely contribute to software evolution, enabling smoother updates for software users.