PORD: a Reversible Debugging Tool using Dynamic Binary Translation

Xiyang Liu, Tao Liu, Zhiwen Bai, Yan Wang, Haoying Mu, Chunxiang Li
Software Engineering Institute, Xidian University, China
xiyangliu@gmail.com, fliuoxdu, baizhiwen, ywangd, haoyingmu, julia.lcxg@gmail.com

With the growing complexity of modern software systems, there is an urgent requirement for a reversible debugger that can execute program backward to historical points. A reversible debugger enables programmers to speed up the location of the failure of programs and improve the efficiency of debugging process dramatically.

This paper presents a prototype of a reversible debugger referred to as PORD. PORD employs an instrumentation approach of dynamic binary translation to save the execution context of the program to make preparation for the reverse execution. PORD consists of two parts: a virtual machine and a remote control interface. We implement the virtual machine on the basis of QEmu's user mode emulator on Linux/x86 platform. The virtual CPU of the virtual machine behaves as a translator to perform dynamic binary translation that refers to the process of converting the source binary program to the target binary program that will be executed on the native CPU directly. During the translation process, the code to record the execution context will also be inserted into the translated code. All translated code is cached for being reused again. The remote control interface is implemented on the basis of GNU Debugger by introducing some reverse execution commands. The two parts communicate by the gdb remote debugging protocol.

PORD provides two modes in reverse execution: binary translate mode and binary copy mode. Binary-translate mode is applied to the case that the guest application and the host platform possess different instruction set architectures. This mode enables the cross-platform reverse debugging, which can greatly facilitate software development of embedded systems. When the guest application and the host platform have the same instruction set architecture, the binary-copy mode is employed. This mode optimizes the translation process in binary-translate mode by copying the source code into the translated code cache directly. The speed of debugging programs with PORD is reasonable, especially for the binary-copy mode which has a near-native speed.

Non-Functional Requirements Elicitation and Incorporation into Functional Models

Xiaoyu Song
Institute of Computing, Theory and Technology, Xidian University, China
yming.song@gmail.com

Top-quality software architecture should deal with both functional and non-functional aspects of the systems. Most researches so far have been directed primarily at the functional, regardless of the non-functional which has a great impact on software quality and development efficiency. This paper proposes a new strategy on how to elicit non-functional requirements from requirement documents and how to automatically incorporate non-functional requirements to the functional models. We devise a framework for formalizing non-functional requirements, and adopt the UML models as the function modeling representation. In order to validate our strategy, we applied it to a case study, a credit card system. The results of the case study convince us that the systems with non-functional requirements considered have higher quality and less expenditure.

Validating Software Reliability by Binary Translation

Yi Wang
Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China
yi_wang@sjtu.edu.cn

Virtual execution environment (VEE) is becoming increasingly popular these days. But how to validate the reliability of software in such environment is not well studied yet. The mechanism of binary translation, which is an important approach to VEE, provides opportunities for monitoring and validating software reliability by means of instrumentation. This poster proposes a framework for validating software reliability in terms of binary translation technology. It uses IR level instrumentation is the core components for performing validation. In the poster, we briefly describe this framework, and provide three common instrumentations can be used in validating software reliability. These are memory tracing, address checking, and data flow analysis. Detailed information is also included.