Analysis of Parallel C++ Programs

Vladimír Štill

Abstract

Parallel software offers a promise of full utilisation of modern hardware. Unfortunately, building a parallel program presents some additional challenges for the programmers. In this thesis, we introduce some improvements to the analysis of parallel C++ programs. In particular, we aim to help with the discovery of hard-to-find bugs.

As our first contribution, we deal with some of the problems related to analysis of high-level programming languages, including their advanced features and standard libraries. We consider this topic important as comprehensive language support makes the analysis tool more usable by programmers in practice. Comprehensive language support is not an easy task. However, we show it is still manageable with the right combination of reuse of existing execution-oriented components and design of new, verification-oriented ones. In this work, we deal with C++ support for the DIVINE verifier in general, and its support for C++ exceptions in particular.

Our second contribution is a novel approach to the analysis of programs running under the relaxed memory model of Intel and AMD x86 processors. These processors can delay memory stores after independent loads which can lead to peculiar behaviour of parallel programs. This behaviour is often hard to keep track by programmers and therefore can be a source of subtle bugs. We propose a novel way in which the verifier can simulate relaxed memory. Our method aims to minimise introduced nondeterminism and therefore increase the overall performance of the verification.

As our last contribution, we introduce a method for checking local nontermination of parallel programs, i.e., for detection of sections of the program that are supposed to terminate but do not. The local nontermination can be used to detect problems in programs that do not terminate but have parts that must do so (e.g., a server might have a function for handling a request, and this function must always terminate). Our method uses lightweight annotations to mark parts of the program which must terminate. It is not limited to the use of particular synchronisation primitives, but it is only complete for programs which have finite state space (such programs can have infinite behaviour in which specific state is repeated infinitely). Even under this limitation, we believe this technique can help with the design of high-performance parallel algorithms and data structures.

The contributions presented in this thesis are implemented in an open-source software model checker DIVINE and are accompanied by experimental evaluation.