Automatic Extraction of Program Models for Formal Software Verification

PEDRO DE CARVALHO GOMES

Academic Dissertation which, with due permission of the KTH Royal Institute of Technology, is submitted for public defence for the Degree of Doctor of Philosophy on Friday the 4th December 2015, at 2:00 p.m. in F3, Lindstedtsvägen 26, KTH, Stockholm.
Abstract

In this thesis we present a study of the generation of abstract program models from programs in real-world programming languages that are employed in the formal verification of software. The thesis is divided into three parts, which cover distinct types of software systems, programming languages, verification scenarios, program models and properties. The first part presents an algorithm for the extraction of control flow graphs from sequential Java bytecode programs. The graphs are tailored for a compositional technique for the verification of temporal control flow safety properties. We prove that the extracted models soundly over-approximate the program behaviour w.r.t. sequences of method invocations and exceptions. Therefore, the properties that are established with the compositional technique over the control flow graphs also hold for the programs. We implement the algorithm as ConFlEx, and evaluate the tool on a number of test cases. The second part presents a technique to generate program models from incomplete software systems, i.e., programs where the implementation of at least one of the components is not available. We first define a framework to represent incomplete Java bytecode programs, and extend the algorithm presented in the first part to handle missing code. Then, we introduce refinement rules, i.e., conditions for instantiating the missing code, and prove that the rules preserve properties established over control flow graphs extracted from incomplete programs. We have extended ConFlEx to support the new definitions, and re-evaluate the tool, now over test cases of incomplete programs. The third part addresses the verification of multithreaded programs. We present a technique to prove the following property of synchronization with condition variables: "If every thread synchronizing under the same condition variables eventually enters its synchronization block, then every thread will eventually exit the synchronization". To support the verification, we first propose SyncTask, a simple intermediate language for specifying synchronized parallel computations. Then, we propose an annotation language for Java programs to assist the automatic extraction of SyncTask programs, and show that, for correctly annotated programs, the above-mentioned property holds if and only if the corresponding SyncTask program terminates. We reduce the termination problem into a reachability problem on Coloured Petri Nets. We define an algorithm to extract nets from SyncTask programs, and show that a program terminates if and only if its corresponding net always reaches a particular set of dead configurations. The extraction of SyncTask programs and their translation into Petri nets is implemented as the STaVe tool. We evaluate the technique by feeding annotated Java programs to STaVe, then verifying the extracted nets with a standard Coloured Petri Net analysis tool.
Software Verification, Static Analysis, Program Models, Petri Nets, Compositional Verification, Concurrency