

Verify Properties of Mobile Code

Songtao Xia
sxia@cse.ogi.edu

September 20, 2001

Abstract

Given a program and a specification, you may want to verify mechanically and efficiently that this program satisfies the specification. Software verification techniques typically involve theorem proving. If a formal specification is easily available, consumption of computational resources is a major issue. Mean while, we shall not overlook the psychological factors. Often, you need extra expertise to verify a program. Tools that can automatically verify programs are helpful.

On the other hand, ubiquitous computing has made the correctness of a program both a security and a performance issue. If you run a piece of mobile code on your machine, you will expect that the code does not access storages unlawfully. To make sure bad things won't happen, performance is sacrificed. If programs are written in an intermediate language that is able to capture and verify properties mentioned above, your host machine will benefit from it.

My research focuses on providing a type-theoretic solution to the verification of mobile programs. One of our primary tools is index types. Index types are a form of non-traditional types. An index type system extends the type system of a language with indices and predicates on those indices. Index types can express properties of program. To type check a program annotated with index types, we often will call an external decision procedure. Another concept used is the proof-carrying code. One of the major advantages of proof-carrying code is that a lot of theorem proving is shifted off-line. When we use proof-carrying code to verify a property, the time spent on verification is mainly on proof-checking, which is considerably cheaper than theorem proving.

We first started working on static array bounds checking problem on index-typed Java bytecode. The goal is to for a code consumer to discover statically that some array accesses are safe and can relief the running time checking. We understand that static array bounds analysis is not decidable. We will stick to run-time check if static checking fails to give an affirmative answer.

In the index-typed bytecode and in proof-carrying code in general, the properties concerned are mainly about a state, instead of a path. For two reasons we are interested in the properties on paths and model-checking. First, data flow analysis problems can be represented as a model-checking problem on an abstracted control-flow diagram. This approach allows us to use model-checking as a generic algorithms to analyze a program. Second, certain temporal properties, such as the deadlock freedom, are traditionally verified by model-checking. Model-checking is computational expensive due to space explosion problem. Using abstraction can reduce the state space so that we can model-check an abstract state space quicker than checking concrete state space. Computation of an abstraction, however, takes large percent of time in the verification process. Thus, if we establish a valid (and useful) abstraction, we can speed up the consequent model-checking even to such a level that the verification can be done on-the-fly, something not achieved before. When this note is written, we are working on a framework that utilizes proof-carrying code to assist model-checking at a client machine. We also plan to use a type system to formalize the timing requirement of a piece of code.