Automated theorem proving algorithms

QRA
critical thinking

June 30, 2018

As technical systems have become more and more complex, verifying that a design works as intended has become an increasingly difficult task. Traditionally, the testing of such systems has relied on model simulations. In this approach a model of the system is fed a wide range of input values, and the results compared against specifications to verify that the system is behaving as expected. Although effective at identifying certain kinds of bugs, this method of testing can easily miss more subtle errors – if a bug requires a particular set of unlikely conditions to be met before causing any problems, for instance, simulation-based testing won't find it unless the exact set of inputs needed to trigger those conditions are fed to the model.

QRA is at the forefront of an entirely new approach to systems testing: the use of formal methods to mathematically verify that a system design is correct. In formal methods testing, the system is treated as a set of mathematical statements, and programs such as Satisfiability Modulo Theories (SMT) solvers are used to attempt to prove that certain properties of the statements hold. The advantage of using formal methods in this manner is that conclusive guarantees about the behaviour of the system can be obtained: we can confirm that desired or expected behaviour is mathematically certain to happen, or that unexpected or non-desired behaviour is mathematically impossible. The nature of the proofs obtained in formal methods testing means that the entire space of possible inputs to the model is covered, including strange edge cases that simulation-based testing might miss. Moreover, in cases where we find that the expected behaviour isn't guaranteed to hold - in cases where there is a bug, in other words - we can provide specific counterexamples, demonstrating a set of inputs for which the unexpected behaviour occurs. This can be invaluable for systems engineers trying to hone in on the source of a problem in their design.

Although formal methods testing can be a powerful tool, it does have limitations. Despite significant recent advances in SMT solvers, there are still many problems that they are unable to solve. As a result, in many cases when a particular query about a model is posed to our software, we are unable to return a result, and it is not always clear why. In some cases the model is simply too big, and a result would be returned if the analysis were allowed to continue long enough. In other cases the query is posed in a way that is ill-suited to the provers and solvers we use, and another equivalent formulation of the query might yield a result instantly. In still other cases the query (or the model) falls outside of the theories that the solvers can handle entirely.

The goal of this problem is to help further our understanding of the circumstances under which our software will be unable analyze a query about a model, and why. To provide initial direction, we will provide two complex models which our software can partially, but not completely analyze. You will explore what can and cannot currently be proved about these models with our software, and for the cases where proofs can't be obtained, investigate why not. This might involve working with the models themselves (e.g. stripping down the models until they're simple enough to be completely analyzed, or looking at the scaling behaviour of analysis times as model complexity/size is increased or decreased). Or it might involve working on the SMT solver side of things, looking at whether or not different formulations of the problem might allow for analysis, or if different tactics and strategies that the solver can try might be helpful.

A desirable outcome from this investigation would at least include: 1) a more comprehensive mapping than we currently have of the boundary between what is and is not proveable about these models with our software, and more information about scaling behaviour. It might also include 2) pushing that boundary forwards, through whatever means you can think of, allowing us to prove more about the models than we currently can. In addition, a secondary goal of the investigation would be to gain an understanding of the *general* characteristics (extending beyond just the two examples provided) that are likely to make a model difficult to analyze. For instance, how much does the topology of a model matter, over and above the effect of its size? Are "wide" models more or less challenging than "deep" models? Are certain model components particularly troublesome? Information like this could be very valuable to QRA, going forward.

Relatively little mathematical background is required to work on this problem, although students familiar with mathematical logic or theoretical computer science and algorithms might find it particularly interesting. Some familiarity with computer programming could be an asset, but is by no means required.