Soundness and its Role in Bug Detection Systems

Yichen Xie Mayur Naik Brian Hackett Alex Aiken

Computer Science Department
Stanford University
Stanford, CA 94305

The term *soundness* originated in mathematical logic: a deductive system is sound with respect to a semantics if it only proves valid arguments. This concept naturally extends to the context of optimizing compilers, where static analysis techniques were first employed. There, soundness means the preservation of program semantics, which is the principal requirement of a correct compiler.

In bug detection systems, soundness means the ability to detect *all* possible errors of a certain class. Soundness is a primary focus of many proposals for bug detection tools. Tools that do not offer such guarantees are sometimes summarily dismissed as being, well, unsound, without regard to the tool's effectiveness.

However, soundness has costs. Beyond the simplest properties, analysis problems are often statically undecidable and must be approximated conservatively. These approximations may be expensive to compute or so coarse that a substantial burden is imposed on the user in analyzing spurious warnings (so-called *false positives*) from the tool.

In our view, successful approaches to bug detection (and program analysis in general) balance three desirable, but often competing, costs: soundness, or rather, the cost of false negatives that result from being unsound; computational cost; and usability, as measured in the total time investment imposed on a user. The question "*Is soundness good or bad?*" does not make sense by itself. Rather, it can be discussed only in the context of particular applications where these costs can be estimated.

This simple, "three cost" model allows us to make a few observations and predictions about the future of bug detection tools.

First, we can expect in almost every application area that widely used unsound tools will precede sound ones, because imposing soundness as a requirement constrains the design space sufficiently that it is simply more difficult and time consuming to find design points that give acceptable usability and computational cost. A good historical example is static type systems, which can be regarded as the canonical example of an analysis where soundness is very desirable. The most widely used languages of the 1980's (C and C++) deliberately had unsound type systems because of perceived usability and performance problems with completely sound type systems. The first widely used language with a static type system intended to be sound (Java) did not appear until the mid-1990's. As another example, the first successful checkers for concurrency problems used dynamic analysis techniques. Because they are dependent on test case coverage, dynamic analyses are unsound almost by definition. Today there is still no completely satisfactory static checker for concurrency errors for any mainstream programming language. We also note that sound systems are sometimes used in an unsound way in practice by, for example, turning off global alias analysis to reduce false positives.

Second, while sound systems will be slower to appear, they will appear. One can expect widely used sound analyses in two different classes of applications. In situations where the extra cost of soundness is minimal there will be no reason not to be sound. In applications where the cost of a single missed bug is potentially catastrophic, users will be more willing to sacrifice usability and performance for soundness. Areas such as security critical, safety critical, and mission critical applications are all likely targets for sound analyses. In general, however, unless a sound analysis can approach the performance and usability of unsound bug finding tools, the sound analysis will be used mainly in applications where the potential cost of unsoundness is highest.

Third, we expect most sound systems will assume at least some user annotations. As mentioned above, most of the analysis problems of practical interest are undecidable, and it is usually impossible to compute both a sound and reasonably precise (i.e., nearly complete) analysis for such problems. However, it is a striking property of many analysis applications that a small amount of additional information dramatically lowers the cost from undecidable to, say, linear time. This phenomenon has been studied extensively in the area of type theory, where the enormous difference in many type systems between the computational complexity of type inference (i.e., analysis with no annotations) and of type checking (i.e., analysis with user annotations) is well known. A little extra information can make many very hard analysis problems quite straightforward. Despite the understandable reluctance to impose any extra work, no matter how small, on users, we expect in most cases adding annotations will be the best path to a practical sound system.