In Defense of Unsoundness

Ben Livshits, Manu Sridharan, Yannis Smaragdakis, and Ondřej Lhoták
Microsoft Research, IBM Research, University of Athens, University of Waterloo

Tens if not hundreds of static program analysis papers are published every year, many of them at top venues such as PLDI and POPL. Are these analyses sound? That is, do they truly capture all possible program behavior and produce results that have to hold during program execution? The programming language research community often assumes so. Yet the answer to this question is a resounding "no"! Virtually all recent whole program analyses for realistic languages are unsound and omit conservative handling for program features that are in common use.

Our message here is threefold:

- First, we bring forward the ubiquity of unsound assumptions in static program analysis. For static analysis experts, this is no surprise. For the rest of the community, which expects to use analyses as a black box, this unsoundness is far less understood, particularly since in published papers, unsoundness often lurks in the shadows, with caveats only mentioned in an off-hand manner in an implementation or evaluation section.
- Second, we argue that soundness is extremely hard to achieve for a whole-program analysis in a realistic, modern language, due to programming language features that are very hard or even impossible to analyze precisely. Therefore, unsoundness of some kind is almost a necessity. Analyses that model tricky language features conservatively end up losing scalability and precision.
- Third, we draw a distinction between analyses that offer best-effort soundness vs. analyses that embrace unsoundness because their clients can tolerate it. We use the term *soundy* for analyses that aim for soundness and only give up when such soundness would be truly detrimental for other key analysis properties. The term arose in a recent Dagstuhl seminar on pointer analysis, where participant after participant found the need to classify their analysis as either "not aiming for soundness, just usefulness" or "sound, except for the usual caveats we all know about", i.e., *soundy*. Both kinds of analyses are unsound, but "unsound" is useless as a classifier, since, as we argue, unsoundness is inevitable.

We believe that the community should both embrace unsound analyses and establish a better understanding of how such analyses should be evaluated. These steps will lead to a better vetting of soundy techniques that are already being published, and to the publication of novel, useful techniques that reject soundness from the start. It will also create a shared understanding of what it takes to produce analyses that can be used in practical tools, bringing together the more theoretical and practical sides of the programming languages research community.

Unsoundness: Inevitable but Desirable?

The typical (published) whole-program analysis extolls its scalability virtues and briefly mentions its soundness caveats. For instance, an analysis for Java will typically mention that reflection is handled "as in past work", while dynamic loading will be (silently) assumed away, as will be any behavior of opaque, non-analyzed code (mainly native code) that may violate the analysis assumptions. Similar "standard assumptions" hold for other languages: Many analyses for C and C++ do not support casting into pointers, and most ignore complex features like `setjmp`/`longjmp`. For JavaScript, the list of caveats grows even longer, to include the `with` construct, dynamically-computed fields (called properties), as well as the notorious `eval` construct.

Can these language features be ignored without significant consequence? Realistically, most of the time the answer is no. These language features are nearly ubiquitous in practice. "Assuming the features away" excludes the majority of input programs. For example, very few JavaScript programs larger than a certain size omit at least occasional calls to `eval`.

Could all these features be modeled soundly? In principle, yes. In practice, however, this modeling usually destroys the precision of the analysis, as usually a sound modeling must be highly over-approximate. This imprecision in turn destroys scalability, as heavyweight analysis techniques end up computing a huge result that is so imprecise as to be useless.

Soundness is not even *necessary* for most modern analysis applications, however, as many clients can tolerate unsoundness. Such clients include IDEs (auto-complete systems, code navigation), security analyses, general purpose bug detectors (as opposed to program verifiers), etc. Even automated refactoring tools that perform code transformation are unsound in practice (especially when concurrency is considered), and yet they are still quite useful and implemented in most IDEs.

Notably, other research communities (software engineering, operating systems, computer security) have been highly receptive of program analyses that are explicitly unsound yet useful. The PL community has been more dogmatic in its interpretation of soundness claims, resulting in a mismatch of understanding between analysis experts (who recognize that unsoundness is inevitable) and other PL experts who are not aware of the typical caveats and environment conditions.

Soundiness

This paper introduces the term *soundiness* as a shorthand for "soundness, except for the usual caveats that experts know about and consider inevitable." Soundiness is in fact what is meant in many papers that claim to be sound. Despite the tongue-in-cheek flavor of the term, we view soundiness as taking a principled, best-effort approach to faithfully modeling program semantics. In other words, program behaviors are modeled soundly, except where it would compromise the scalability and precision of the analysis to do so.

We draw a distinction between a soundy analysis, which aims to capture all dynamic behaviors within reason, and an unsound analysis that deliberately ignores certain behaviors (though we note that there is nothing wrong with the latter, if the analysis is useful for its intended clients). We argue that soundiness is a good line in the sand to draw in order to avoid abuse of the observation that "everyone's analysis is unsound." The precise definition of soundiness will vary per language, but, for instance, an analysis cannot claim to be soundy for Java when it ignores entirely the behavior of the standard library, since the latter is not part of the well-understood soundness lapses in the community.

Evaluating Unsound Analyses

Unsoundness levels must be distinguished since *these choices have a huge impact on analysis results*, both in terms of scalability and precision. For instance, equivalent published analyses of Java programs report numbers of reachable methods that differ by a factor of *two*, entirely due to modeling different language features. Even worse, analyses that assume away some language features are often evaluated using programs that *violate* these assumptions, casting doubt on what can be learned from the results. For example, some Java benchmarks use reflection in initialization code to load the main part of the program, so an analysis that ignores reflection will ignore the bulk of the actual benchmark.

Clearly, an improved evaluation methodology is required for these unsound analyses, to increase comparability of different techniques. While further work is required to devise such a methodology in full, we believe that at the least, some effort should be made in experimental evaluations to compare results of an unsound analysis with observable dynamic behaviors, as a sanity test of whether important behaviors are being captured.

Moving Forward

We advocate the following:

- The PL research community should embrace unsound analysis techniques and tune its soundness expectations. Soundy is the new sound, de facto, given the research literature of the past decades.
- Papers on unsound analyses should both explain the general implications of their unsoundness and also evaluate the implications for the benchmarks being analyzed.
- For language features that are most often ignored in unsound analyses (reflection, `setjmp`/`longjmp`, `eval`, etc.), more studies should be published to characterize how extensively these features are used in typical programs and how ignoring these features could affect standard program analysis clients.
- As a community, we should provide guidelines on how to write unsound analysis papers, perhaps varying per input language, emphasizing which features to consider handling --- or not handling.