Object Structure Analysis: Everything You Always Wanted to Know About Your Program

Abstract

This will be a description and hands-on demo of work in progress: an automatic program analyzer producing precise answers to questions about run-time object structures.

During its executions, a program can produce huge and complex data structures. Can we confidently predict some of their properties? Examples, important for program verification and compiler optimization, include:

- Aliasing: can two pointer expressions, say x and a.b.c, ever point to the same object?
- Reachability: is there a path from a given object to another?
- Void safety: in x.f, can x ever be a null pointer, causing the program to crash?

The last example is critical: void safety remains in almost all languages an unsolved problem. (The principal exception is Eiffel, which achieves void-safety by design.) Null-pointer calls are unpredictable and can crash any program, even after months of flawless operation. The Common Vulnerabilities and Exposure security database is replete with catastrophic null-pointer attacks involving many widely used products, even the JPEG protocol.

To provide a widely applicable solution, I have developed a theory of object structures, duality semantics, and implemented it. I will briefly outline the theory (based on abstract interpretation and my own earlier work on the Alias Calculus); but mostly the talk is a demonstration of the resulting tool, in its current partial state, on a number of examples. The target programming language is a concentrate of the mechanisms of major languages, making the concepts applicable to C, Java, C# etc.

Anyone who has had even a superficial brush with a static analysis tool, starting with the old “lint” for C, knows that in the end only one thing counts: removing false alarms. With most tools, after the initial thrill of learning a thing or two that they did not know about their program, users quickly lose interest. The reason is that these tools produce a deluge of irrelevant warnings; users must then manually and painfully comb through the list, to weed out the few genuine risks. Precision (minimizing the rate of false alarms) is the name of the game. The requirements are cruel: the only acceptable rate of false alarms, if a tool is to have any chance of practical acceptability, is very close to zero.

Thanks to the theory, the tool presented (in addition to, I hope, being sound) is precise in its approximation of run-time properties. That precision is tunable: it depends on the depth of analysis, which the user can specify, a better precision yielding a longer computation. For every context – from immediate feedback during interactive development to final thorough check before a major release – a user can decide the appropriate tradeoff between computation time and precision.

I will present the state of the work, its achievements and limitations, and say a few words about my development process.

Speaker: Bertrand Meyer is Provost and Professor of Software Engineering at the newly created Schaffhausen Institute of Technology in Switzerland. He was previously a professor and head of department at ETH Zurich (Swiss Federal Institute of Technology) and hold associated positions at Innopolis University and University of Toulouse. He is CTO and co-founder of Eiffel Software, based in Santa Barbara (CA). He is the author of a number of well-known books on various aspects of software engineering, most recently “Agile!”, a tutorial and critical analysis of agile methods, and earlier the best-seller “Object-Oriented Software Construction”.