DOI:10.1145/1995376.1995399

Technical Perspective
Abstracting Abstract Machines

By Olivier Danvy and Jan Midtgaard

THE GOAL OF program analysis is to statically predict runtime properties of programs without running them. The semantic approach to program analysis originates in Cousot's path-breaking work on abstract interpretation: start from a formal mathematical model of program execution—a *semantics*—and approximate it with Galois connections (or similar means) into a computable model based on lattices of runtime properties that accounts for all possible execution paths. Each program gives rise to a collection of equations that are then typically solved by fixed-point iteration.

Semantics-based program analysis therefore requires one to (1) start from a "friendly" semantics; design a "congenial" lattice of runtime properties; (3) associate a "relevant" set of equations to a program; and (4) solve these equations efficiently.

Each of these requirements is fraught with difficulties:

1. Among the varieties of formal semantics that exist (operational, denotational, axiomatic, among others) and their sub-varieties (for example, small step or big step), where is your friendly semantics? Ideally, it should lend itself to a good approximation into a computable model.

2. What is a congenial lattice of runtime properties? How wide should it be? How high? Ideally, it should lend itself to a good widening operator that accelerates the convergence of fixed-point iteration without compromising the precision of its result.

3. What is a relevant set of equations? Ideally, each equation should mimic the friendly semantics as closely as possible.

4. What is the best representation of equations and the most efficient way to solve them? This is an algorithmics problem.

Effective answers to each of these questions have been found before, but it is like each of them is a tour de force.

In the following paper, David Van Horn and Matthew Might take a radical bet of simplicity and effectiveness:

▸ Since most semantic artifacts are inter-derivable, without loss of generality, they select abstract machines—deterministic state-transition systems with potentially infinite state spaces—as their friendly semantics.

We find Van Horn and Might's scientific contribution to be an effective tutorial on how to develop a higher-order program analysis by abstracting an abstract machine.

▸ They then refactor each abstract machine into a non-deterministic state-transition system with a finite state space.

Their methodology is concretely useful: it enables program-analysis designers to start from an existing abstract machine rather than from an ad hoc, tailored one, and then factor it uniformly into an abstraction-friendly semantic artifact. Their methodology is effective: it scales to a variety of computational situations involving realistic programming-language constructs, for example, exceptions. Their methodology is structural and generic: it enables program-analysis designers to concentrate on what is specific to their analysis and is still difficult—their lattice of runtime properties, their widening operator, how to represent their equations, and how to solve them efficiently—instead of being forced to perform one global tour de force after another, from scratch, every time.

As such, we find Van Horn and Might's scientific contribution to be a significant stepping stone conceptually and practically as well as an effective tutorial on how to develop a higher-order program analysis by abstracting an abstract machine. We also found their article a pleasure to read. C

Olivier Danvy (danvy@cs.au.dk) is an associate professor and **Jan Midtgaard** (jmi@cs.au.dk) is a post-doctoral researcher in the Department of Computer Science at Aarhus University, Aarhus, Denmark.