From Principles to Programming Languages (and Back)

Shriram Krishnamurthi

Computer Science Department, Brown University
sk@cs.brown.edu

Categories and Subject Descriptors D.3.1 [*Programming Languages*]: Formal Definitions and Theory – semantics; D.2.4 [*Software Engineering*]: Software/Program Verification

Keywords semantics engineering, democracy of languages

The Democracy of Programming Languages

While budding linguists have always had the freedom to tinker, the Web has revolutionized linguistic experiments in two key ways. First, the language-neutrality of network abstractions have made it possible for Web-deployed applications to be written in any language at all, allowing new languages to show off their capabilities. Second, the Web has greatly facilitated the dissemination of these experimental languages. As a result, new languages crop up almost daily, lovingly tended by their designers and user communities and vigorously debated on popular forums. This is a far cry from the programming languages world of just over a decade ago.

Most of these languages are not designed and developed in formal settings. As a result, they lack many of the tools that this conference's audience takes for granted: static analyses, type systems, verification engines, and so on. As we know, these tools not only improve usability, their construction serves as a design check: odd spots in the language design usually manifest themselves as difficulties in building or proving properties about the tool. This leads to a virtuous co-development cycle, or at least forces the language designer to justify the inclusion of such difficult elements.

A cynic might question why every language experiment deserves these tools. Because many of these languages will never see widespread adoption, they may not be worth a significant investment of formal effort. To this, there are three counter-arguments:

1. A language may be intended for in-house use or only to solve a particularly vexing problem in some important system. Thus, the size of its user-base is not a good measure of significance.
2. By the time a language becomes popular, it often already has frameworks, applications, or both, and hence already needs these tools.
3. Because tools are often vital to convincing users to employ the language—all the more given the physical distance between designer and user that the Web enables—the absence of such tools means promising language design experiments may never see the light of day.

POPL'13, January 23–25, 2013, Rome, Italy.
ACM 978-1-4503-1832-7/13/01.

It is therefore worth considering a research program that simultaneously serves the democracy of languages while upholding the values we hold dear, such as sound tools that are backed by theorems about their properties.

The Challenges of *Programming* Languages

Languages that developers use to write non-trivial programs rarely resemble the elegant cores we formalize. To have direct impact, three *semantics engineering* questions deserve special attention:

1. Beyond natural language specifications and de-facto normative implementations, language designers increasingly provide conformance test suites. These suites are formal objects (unlike natural language specifications), probably thoughtfully decomposed (unlike many implementations), and kept current (unlike many specifications). How can we ease the transition from conformance suites to formal semantics that match them?
2. Programs are increasingly a rich mix of code in a base language and code written atop frameworks. How can we formulate a semantics for the implicit languages of these APIs? How do we determine the invariants of these languages, to check that base language programs don't violate them?
3. In practice, programs in one language often depend on programs from other languages. While it is tempting to union the semantics of all these languages to reason about such programs, this approach usually leads to an intractable mess of details. Can we define a family of semantics so that we can "zoom" in to the appropriate level of detail? Can such a family accommodate both sound and tractable cross-language reasoning?

Going beyond core languages means also focusing on the needs of the users who employ them. Because programming languages are a developer's primary human-computer interface, the human writing the programs deserves as much attention as the language. In principle, in a complex system with mutual dependence, the two components must be co-designed; but when one of those components is a human, change can usually only be achieved at evolutionary pace. Programming languages, and their environments, therefore need to better account for the established cognitive abilities and disabilities of their users.

The democracy of languages means the future for programming language research is brighter than ever. Furthermore, linguistic thinking applies broadly and is in urgent need. My own group has spent the past decade applying a linguistic mindset—and generating some core results in return—not only to traditional programming and specification languages, but also to industrial access-control languages, Web servers, Web browsers, Excel spreadsheets, Internet routers and firewalls, social networks, and Alice-and-Bob cryptographic diagrams. In all these areas, and more, there is a great need for the clarity and focus on primitives and composition that is characteristic of programming languages thought.