Do Abstract Machines Make Abstract Heat?

Benjamin Schulz

December 15, 2011

Symbolic computation occupies its own universe with its own laws. Intuitively, such abstract and purely symbolic universes are also extremely orderly. Given this order, it seems only natural that formal logic should provide a powerful tool for reasoning about phenomena in the symbolic universes of computation. This is no idle observation; nearly a century of intellectual labor has developed and analyzed the deep and remarkable relation between logic and computation, yielding useful tools in the form of type systems, theorem provers, model checkers, and novel programming language paradigms. Logic is an essential tool for computer science because it makes precise and clear the ways in which programs are, in their essence, simple and orderly.

In a precise and very well-defined sense, programs are also fundamentally complex and disorderly. This fact should not be completely surprising, or even wholly controversial; there are well-known theoretical limits to what logic can reveal about programs. It is in the face of exactly such complexity and disorder that logic is powerless. The implications of this observation, however, are profound; the pioneering work of Kolmogorov, Solomonoff, and Chaitin illustrated that even purely symbolic, computational universes are subject to the same tendency to complexity and disorder observed in the physical universe. Just as the study of dynamic physical systems promises a deeper understanding of phenomena that heretofore confounded the physical sciences, a deeper exploration of complexity offers the potential for new and powerful tools for analyzing and understanding computational systems.

Unfortunately, most developments in computational complexity have been confined to pure mathematics and to the theory of abstract computation. Tools for precisely analyzing the structure and degree of program complexity, however, would find application in virtually all fields of computer science, and lay the foundations of a methodology that would solidifiy the position of software development as a genuine engineering practice. Programming language research, with its awareness of the breadth and diversity of computational paradigms, its dynamic synthesis of theoretical and practical aspects of computing, and its mastery of program analysis and transformation, is uniquely positioned to realize the theoretical consequences of program complexity as practical tools. In order for these results to take shape, however, researchers must widen their view to encompass both halves of the dual nature of computation, as something essentially simple in structure but inescapably complex in its consequences.