Concurrent Semantics Without the Notions of State or State Transitions

Edward A. Lee

University of California, Berkeley
eal@eecs.berkeley.edu

Abstract. This paper argues that basing the semantics of concurrent systems on the notions of state and state transitions is neither advisable nor necessary. The tendency to do this is deeply rooted in our notions of computation, but these roots have proved problematic in concurrent software in general, where they have led to such poor programming practice as threads. I review approaches (some of which have been around for some time) to the semantics of concurrent programs that rely on neither state nor state transitions. Specifically, these approaches rely on a broadened notion of computation consisting of interacting components. The semantics of a concurrent compositions of such components generally reduces to a fixed point problem. Two families of fixed point problems have emerged, one based on metric spaces and their generalizations, and the other based on domain theories. The purpose of this paper is to argue for these approaches over those based on transition systems, which require the notion of state.

1 Introduction

In this paper, I argue that basing the semantics of concurrent systems on the notion of state and state transitions is problematic. The resulting models often have unnecessary nondeterminism in the sense that the nondeterminism is an accident of the choice of modeling technique, rather than an intrinsic or interesting property of the system under study. This complicates analysis and impedes understanding. Moreover, such models fail to be compositional, in the sense that deterministic transition systems, when composed, often become nondeterministic, and that this nondeterminism reveals few if any insights about the system. The result is poor descriptions of the composite behavior.

Introducing time into concurrent models can help, but it either reduces the problem to sequential computation or it relies on a fictional abstraction, Newtonian universal time. In Newtonian universal time, every component in a distributed system shares a common notion of time. Networked computational systems have no mechanism for establishing this common notion of time. Considerable effort is required to establish even an approximate common notion of time [26], and since it is necessarily approximate, the resulting models will be either inaccurate or unnecessarily nondeterministic. When a strongly common notion of time is assumed in the semantics, as in for example discrete-event systems, then distributed or parallel execution becomes a major challenge [19,57].

E. Asarin and P. Bouyer (Eds.): FORMATS 2006, LNCS 4202, pp. 18–31, 2006.

There is a strong draw, however, towards using the notions of state and state transitions in semantics. These notions are deeply rooted in our understanding of computation. Programming languages are based on these concepts (and as a consequence, adapt poorly to concurrent computation [32]). Even our foundational notions of semantics are wedded to state. In [55], for example, Winskel explains denotational semantics to be a description of commands as functions mapping a syntax into a function that maps state into state. Winskel observes, in fact, that this notation appears to not be powerful enough for parallelism and fairness (presumably because of its focus on a single global state). When it comes to modeling parallelism, in [55] Winskel focuses on Hoare's CSP [25] and Milner's CCS [42]. He gives both in terms of labeled transition systems, where the labels can include input/output operations. But labeled transition systems are intrinsically based on the notion of state. Coupling two deterministic non-interacting processes (a trivial composition) under either CSP or CCS semantics will yield a nondeterministic semantic model. This nondeterminism contributes nothing to the understanding of the system. It reflects the uninteresting and inconsequential multiplicity of possible interleavings of independent actions.

The focus on transition systems follows naturally from our core imperative notions of computation. There has been, of course, considerable exploration of concurrent alternatives to imperative models of computation. For example, in [20], Goldin et al. describe a persistent Turing machine (PTM), which has three tapes: input, working, and output. It continually processes data from the input, producing outputs. Networks of these can interact. A "universal PTM" can simulate the actions of any PTM. Goldin et al. argue that any "sequential interactive computation" (which has only a causality restriction) can be represented by a PTM. PTM's have stream and actor semantics, and are intrinsically concurrent. But they are not compositional. Consider again two non-interacting PTSs. Without synchronization between these, the resulting composition is not usefully modeled as a PTM. Similarly, in [22], Gössler and Sifakis argue eloquently for a separation of behavior models from interaction models. But "behavior" is again given as transition systems. Once again, without imposing strong synchronization, a composition of behaviors is not usefully modeled as a behavior.

Of course, one can introduce synchronization to alleviate these problems. The synchronous languages [8] take a particularly strong stance on this, where concurrent computations are simultaneous and instantaneous, and at each "tick" of a global "clock" variables have a value given denotationally as a fixed point. This model is clean and compositional, but has proved notoriously difficult to implement in parallel or distributed systems. This has led to a focus on "global asynchronous, locally synchronous" (GALS) models [9]. These are, by definition, not compositional, since compositions of asynchronously interacting components cannot be again made synchronous without introducing unnecessary nondeterminism.

Even our notions of equivalence between systems are deeply connected to the notion of state. In [43], Milner originated the idea of observational equivalence as mutual simulation (and bisimulation). Simulation and bisimulation are relations