Abstract. Existing methodologies for the verification of concurrent systems are effective for reasoning about global properties of small systems. For large systems, these approaches become expensive both in terms of computational and human effort. A compositional verification methodology can reduce the verification effort by allowing global system properties to be derived from local component properties. For this to work, each component must be viewed as an open system interacting with a well-behaved environment. Much of the emphasis in compositional verification has been on the assume-guarantee paradigm where component properties are verified contingent on properties that are assumed of the environment. We highlight an alternate paradigm called lazy composition where the component properties are proved by composing the component with an abstract environment. We present the main ideas underlying lazy composition along with illustrative examples, and contrast it with the assume-guarantee approach. The main advantage of lazy composition is that the proof that one component meets the expectations of the other components, can be delayed till sufficient detail has been added to the design.

1 Introduction

In the last two decades, there has been considerable progress in the verification of concurrent, reactive systems. Much of the research has been devoted to the development of formalisms such as temporal logics [Eme90,Lam94,MP92,CM88] and...
process algebras [Hoa85, Mil80], and verification methods [Bar85, dBdRR90, dBdRR94, Sha93a] based on deduction [Eme90, Lam94, MP92, CM88] and model checking [CES86, Kur93, Hol91]. While these techniques are effective on small examples—mutual exclusion, basic cache consistency algorithms, and simple communication protocols—the difficult problem of scaling these techniques up to large and realistic systems has remained largely unsolved.

Large-scale concurrent systems are usually defined by composing together a number of components or subsystems. The typical verification methods are non-compositional and require a global examination of the entire system. In the deductive approach to verification, this means that a property such as an invariant has to be verified with respect to each transition of all of the components in the system. Verification approaches based on model checking also fail to scale up gracefully since the global state space that has to be explored can grow exponentially in the number of components [GL94]. The purpose of a compositional verification approach is therefore to shift the burden of verification from the global level to the local, component level so that global properties are established by composing together independently verified component properties.

To motivate compositional verification, we can consider a very simple example of an adder component $P$ shown in Figure 1 that adds two input numbers $x$ and $y$ and places the output in $z$. Here $x$, $y$, and $z$ can be program variables, signals, or latches depending on the chosen model of computation. The system containing $P$ as a component might require its output $z$ to be an even number, but obviously $P$ cannot unconditionally guarantee this property of the output $z$. It might be reasonable to assume that the environment always provides odd number inputs at $x$ and $y$, so that with this assumption it is easy to show that the output numbers at $z$ are always even. Only local reasoning in terms of $P$ is needed to establish that $z$ is always even when given odd number inputs at $x$ and $y$.

If, as is shown in Figure 2, $P$ is now composed with another component $Q$ that generates the inputs at $x$ and $y$, then to preserve the property that only even numbers are output at $z$, $Q$ must be shown to output only odd numbers at $x$ and $y$. However, the demonstration that $Q$ provides only odd numbers as outputs at $x$ and $y$ might require assumptions on the inputs taken by $Q$, where $z$ itself might be such an input. If in showing that $Q$ produces odd outputs at $x$ and $y$, one has to assume that the $z$ input is always even, then we have an obvious