Computer-aided verification of delay-insensitive protocols
Hemangee K. Kapoor and Mark B. Josephs
Centre for Concurrent Systems and VLSI
School of CISM, South Bank University
{kapoorhk, josephmb}@sbu.ac.uk

Abstract

The structured, parallel programming language of Delay-Insensitive Sequential Processes (DISP) is convenient for the description of communication protocols in terms of input/output bursts. It has previously been shown that asynchronous logic can be automatically synthesised from such descriptions in two steps: first, by translating DISP programs into Petri nets using the software tool di2pn and, second, by synthesising Boolean equations from Petri nets using the software tool Petrify.

We have been investigating the semantics of the DISP language with the aim of supporting computer-aided verification of DISP programs. In particular, the designer may wish to determine whether syntactically distinct programs, \( P \) and \( Q \), are semantically equivalent in that they represent the same protocol, or whether \( Q \) refines (conforms to) \( P \) in that any circuit that implements \( Q \) must also implement \( P \). Fortunately, there is a body of theoretical work, notably that concerning the related languages of Communicating Sequential Processes and Delay-Insensitive Algebra, upon which we are able to build. For example, it is known that refinement is related to nondeterministic choice in such a way that \( Q \) refines \( P \) if and only if \(( P \lor Q)\) is equivalent to \( P \). Also, there are a number of powerful software tools available that can automate such tasks as term rewriting and theorem proving. We have chosen one such tool, maude (http://maude.cs.uiuc.edu/), with which to prototype our ideas.

Our approach has been to identify a core subset of DISP into which any program can be transformed (reduced) by the systematic application of correctness-preserving rewrite rules (algebraic laws). Reduction eliminates such constructs as nondeterministic choice, sequential composition and parallel composition from programs. We have also defined a further transformation (normalisation) that can be applied to programs after reduction, and is such that two programs are semantically equivalent precisely when they have the same normal form. Thus, to show that \( P \) and \( Q \) are equivalent, one reduces \( P \) to \( P' \) and \( Q \) to \( Q' \), normalises \( P' \) and \( Q' \) to \( P'' \) and \( Q'' \), respectively, and compares \( P'' \) and \( Q'' \).

In our presentation we shall consider some simple DISP programs, describing protocols relevant to asynchronous circuit design, and demonstrate that the term-rewriting tool maude can automatically perform reduction, normalisation and comparison.