Problem Solving in Computer Science · Summer 2007 Tom Henzinger

3 Component and Implementation Semantics

Scribe: Ghid Maatouk 20. March 2007

1 Progress reports

The groups reported on their work on Project 1. Each group came up with a definition of implementation without yet defining an associated notion of correctness. The key questions regarded the intended model of computation.

Group $n + 1$ chose a round-based computation model with interleaving on processors and delayed output. The proposed notion of implementation forbids cycles in components. Additionally, they assume that each pair of processors is connected within one network. This is a very strong restriction that may need to be revised in the future.

Principle Be careful not to make assumptions that are too restrictive. Always ask yourself whether these are necessary.

Les Scribouillards focused on connectivity constraints. They formulated three conditions an implementation shall satisfy: input, output, and internal connectivity. This allowed them to map logical connections in the component to physical connections on the platform, and to handle multiple networks. They did not commit to a specific model of computation.
Hint: You can either work with a specific model of computation, or keep several models of computation in mind and try to obtain results that apply to all of them.

The Indoor Group agreed on a scheduling-based model of computation assuming cycle-free components and no delay on physical wires. An implementation is specified by a schedule that should satisfy certain conditions: coverage, network, dependency, load, and reliability.

The Backbenchers chose an FPGA-based model of computation. Here, reliability should measure the ability to perform a computation within a given delay. However, the given definition of a program then needs to be revised.

2 Semantics of Component and Implementation

We are given a component Φ and a platform s. We intend to define an implementation as a mapping $\mu : F \to 2^P$, and to formulate a notion of correctness to be expressed as $s, \mu \models \Phi$.

Component Semantics

A component semantics specifies how the values of the output variables of a component shall be computed from the values of the input variables.

Towards this, we first need to define the meaning of a task. Two ways of viewing this are suggested.

1. In a purely combinational interpretation, tasks are functions from input values to output values. This allows us to specify only functions which don't have a memory of previous inputs, such as addition.
2. We may think of input and output as a *signal*, that is, a sequence of values over time, and interpret a task as a function from input signals to output signals. This interpretation introduces the notion of a state of a task and allows us to specify a wider range of functions, such as integration.

A component is formed by composing tasks. The next step is to define a semantics for the composition of tasks. Again, we distinguish between two views according to whether delays between tasks are introduced or not.

We consider two types of semantics:

Synchronous Semantics In this view, time is assumed to be discrete and to take values in $\mathbb{N}$. For simplicity, we may assume that input values and output values are also restricted to values in $\mathbb{N}$. Accordingly, a signal is a function from $\mathbb{N}$ to $\mathbb{N}$, which associates to each time unit a value. We can then define a task to be of the form

$$t : [R(t) \to \mathbb{N} \to \mathbb{N}] \to [W(t) \to \mathbb{N} \to \mathbb{N}],$$

where $[R(t) \to \mathbb{N} \to \mathbb{N}]$ means that for each read variable and for each time unit, t takes a value as input, and similarly $[W(t) \to \mathbb{N} \to \mathbb{N}]$ means that for each written variable and for each time unit, t produces a value as output. This interpretation assumes that composition of tasks introduces no delays.

Asynchronous Semantics This view differs from the synchronous semantics by the assumption that a delay of one time unit is introduced for every composition of two tasks. Note that we may account for delays even if we are using a combinational interpretation of tasks.

A component semantics can thus be represented as a function

$$\mathcal{C}(\Phi) : [I \to \mathbb{N} \to \mathbb{N}] \longrightarrow [O \to \mathbb{N} \to \mathbb{N}].$$

Implementation Semantics

An implementation semantics specifies what a component mapped to a platform actually computes. This can be represented as a function of the component, the platform, and the implementation mapping:

$$\mathcal{I}(\Phi, s, \mu) : [I \to \mathbb{N} \to \mathbb{N}] \to [O \to \mathbb{N} \to \mathbb{N}].$$

The assumption that the execution of functions on processors is interleaved or atomic, and the way of mapping a function to one or several processors can be encapsulated within such a representation.

Perhaps the best way to define correctness $s, \mu \models \Phi$ is to start by defining $\mathcal{C}(\Phi)$ and $\mathcal{I}(\Phi, s, \mu)$. One idea would be to view both the component and the mapping of a component to a platform as a specification of a system, then to define reliability starting from some "metric" that measures how far a given system is from being fully reliable.