2 Problem 1: A Reliability Calculus

16. March 2007

Let T be a set of tasks.

Definition of Component

A *component* $\phi = (I, O, \pi, \rho)$ consists of a set I of input variables, a set O of output variables, a program π from I to O, and a map ρ: $O \to [0, 1]$ that assigns to each output variable a desired reliability.

A *program* $\pi = (F, L, R, W, t)$ from I to O consists of a set F of functions, a set L of local variables, a map R: $F \to 2^{I \cup L}$ that assigns to each function a set of read variables, a map W: $F \to 2^{O \cup L}$ that assigns to each function a set of written variables, and a map t: $F \to T$ that assigns to each function a task. We require that no two functions write to the same variable, i.e., for all $f, g \in F$, if $f \neq g$, then $W(f) \cap W(g) = \emptyset$.

Intention: For every output variable $o \in O$, if o depends on the set $I' \subseteq I$ of input variables, and the mean time to failure for each input variable $i \in I$ is m_i, then the implementation of the component ϕ must ensure that the mean time to failure m_o for o satisfies the following reliability constraint:

$$m_o \geq \frac{1}{1 - \rho(o) \cdot \prod_{i \in I'} (1 - \frac{1}{m_i})}$$

Definition of Platform

A *platform* $s = (P, N, C, \delta, \lambda)$ consists of a set P of processors, a set N of networks, a map C: $N \to 2^P$ that assigns to each network a set of connected processors, a map δ: $P \times T \to [0, 1]$ that assigns to each processor and each task a load, and a map λ: $P \to [0, 1]$ that assigns to each processor a failure rate.

Intention: For each processor $p \in P$, the execution of a task $t \in T$ requires fraction $\delta(p, t)$ of every time unit. The arrival of failures at processor p is a Poisson process with rate $\lambda(p)$.

Some Questions

1. Define an *implementation* of a component on a platform. To achieve the desired reliability, a function may be replicated on multiple processors. The implementation is *correct* if it satisfies the reliability constraints of all output variables. Define correctness.

2. How hard is it to check the correctness of an implementation? To find a correct implementation? Are there interesting special cases of components and platforms that can be correctly implemented in polynomial time?

3. Can you refine the notion of component to support a compositional proof rule about correct implementability?

4. Change the semantics of programs from synchronous to dataflow.