Homework No. 1 - Solution

EECS 290N
Spring 2009

Edward A. Lee
EECS Department
University of California at Berkeley
Berkeley, CA 94720, U.S.A.

Due: Friday, February 6, 2009

1 Theory Exercises

In Davey and Priestley:

1.5 Prove that the ordered set Σ^{**} of all binary strings is a **tree** (that is, an ordered set P with $\perp$ such that $\downarrow x$ is a chain for all $x \in P$). For each $u \in \Sigma^{**}$, describe the elements covering u.

Solution. Assume to the contrary that Σ^{**} is not a tree. This means that there is a string $s \in \Sigma^{**}$ such that $\downarrow s$ is not a chain. This means that there are two strings $s_1, s_2 \in \downarrow s$ where s_1 is not a prefix of s_2 and s_2 is not a prefix of s_1. These strings must differ at some finite index i. This contradicts that they are in $\downarrow s$, which means they are both prefixes of s.

The covering elements of a string $u \in \Sigma^{**}$ are the string $u0$ and the string $u1$. $\square$

1.10 Let P and Q be chains. Prove that $P \times Q$ is a chain in the lexicographic order. Give formal definitions of the ordered sets $P \dot{\cup} Q$ and $P \oplus Q$. [Hint: Define appropriate orders on $(\{0\} \times P) \cup (\{1\} \times Q)$.]

Solution. For the first part, we need to show that any two elements $(p, q), (p', q') \in P \times Q$ are comparable. That is, either $(p, q) \leq (p', q')$ or $(p', q') \leq (p, q)$.

If $p = p'$, then $(p, q) \leq (p', q')$ if $q \leq q'$ and otherwise $(p', q') \leq (p, q)$, by definition of the lexicographic order. If $p \neq p'$, then $(p, q) \leq (p', q')$ if $p \leq p'$ and otherwise $(p', q') \leq (p, q)$, again by definition of the lexicographic order. Hence, the two elements are comparable.

Following the hint for the formal definitions, let

$$Y = (\{0\} \times P) \cup (\{1\} \times Q).$$

To define the disjoint union $P \dot{\cup} Q$, we assume this set has the pointwise order and that 0 and 1 are incomparable.

To define the linear sum $P \oplus Q$, we assume this set has a lexicographic order and that $0 < 1$.

In both cases, the disjoint union and the linear sum are sets X with the following order isomorphism to the above set Y,

$$f: X \to Y$$

where

$$f(x) = \begin{cases} (0, x) & \text{if } x \in P \\ (1, x) & \text{if } x \in Q \end{cases}$$

It is easy to show that this is a bijection. □

2 Design Exercises

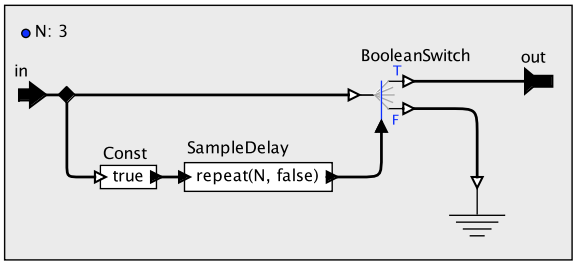
1. As a warmup, create a PN model containing a composite actor with one input port and one output port, where the output sequence is the same as the input sequence except that the first token is missing. That is, the composite actor should discard the first token and then act like an identity function. Demonstrate by some suitable means that your model works as required.

   Augment your model so that the number of initial tokens that are discarded is given by a parameter of the composite actor. It may be useful to know that the expression language includes a built-in function `repeat()`, where, for example,

   ```
   repeat(5, 1) = {1, 1, 1, 1, 1}
   ```

   Note that you can easily explore the expression language by opening an ExpressionEvaluator window, available in the File menu under New. Also, clicking on Help in any parameter editor window will provide documentation for the expression language.

   Solution. The composite actor shown below will do the job:

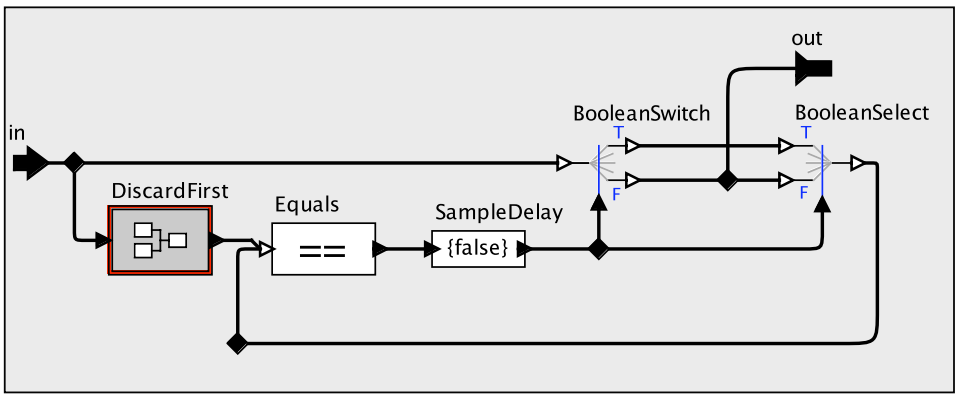
   

   □

2. Create a PN model containing a composite actor with one input port and one output port, where the output sequence is the same as the input sequence except that any consecutive sequence of identical tokens is replaced by just one token with the same value. That is, redundant tokens are removed. Demonstrate by some suitable means that your model works as required.

   Can your implementation run forever with bounded buffers? Give an argument that it can, or explain why there is no implementation that can run with bounded buffers.

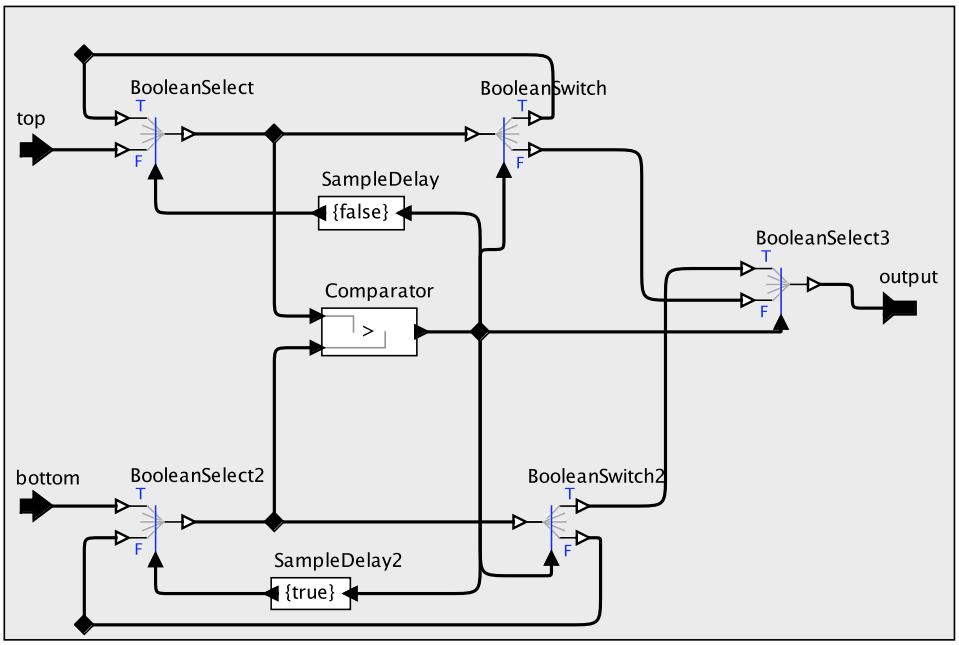
   Solution. The composite actor shown below will do the job:



The DiscardFirst actor is the solution to the previous problem. This runs with bounded buffers. In fact, a buffer of length 1 everywhere is sufficient. The feedback loop will always contain exactly one token, where the first such token is provided by the SampleDelay actor. For every token produced by the SampleDelay actor, exactly one token will be produced by the BooleanSelect actor, and exactly one will be produced by the Equals actor. □

3. Create an implementation of an OrderedMerge actor using only "simple" PN actors. (Note that there is a Java implementation of OrderedMerge in Actors: ProcessNetworks). Your implementation should be a composite actor with two input ports and one output port. Given any two numerically increasing sequences of tokens on the input ports, your actor should merge these sequences into one numerically increasing sequence without losing any tokens. If the two sequences contain tokens that are identical, then the order in which they come out does not matter.

   Solution. The composite actor shown below will do the job:



Each of the two Select-Switch loops keep track of the most recently read token from the corresponding input port. The Comparator thus compares the two most recently seen input tokens and sends the smaller of the two to the output. As a side effect, this triggers another read on the input port from which that token was taken. □

4. In figure 1 is a model that generates a sequence of numbers of the form $2^n 3^m 5^k$ in numerically increasing order, with no redundancies. This model can be found in the PN demos (labeled as OrderedMerge). Can this model run forever with bounded buffers? Why or why not?

   For this problem, assume that the data type being used is unbounded integers, rather than what is actually used, which is 32 bit integers. With 32 bit integers, the numbers will quickly overflow the representable range of the numbers, and wrap around to negative numbers.

   Solution. The model cannot run forever with bounded buffers. The upper left left feedback loop runs with bounded buffers (buffers of length 1 are sufficient). It produces an infinite sequence on its output. That sequence has values $(5, 25, 125, 625, \cdots)$. Every token in this sequence is consumed by the OrderedMerge. Without those tokens, the loop that includes the OrderedMerge would have, at any time, exactly one unconsumed token on some buffer in the loop. But each time one of those tokens is injected into the loop, the loop acquires one additional unconsumed token. □

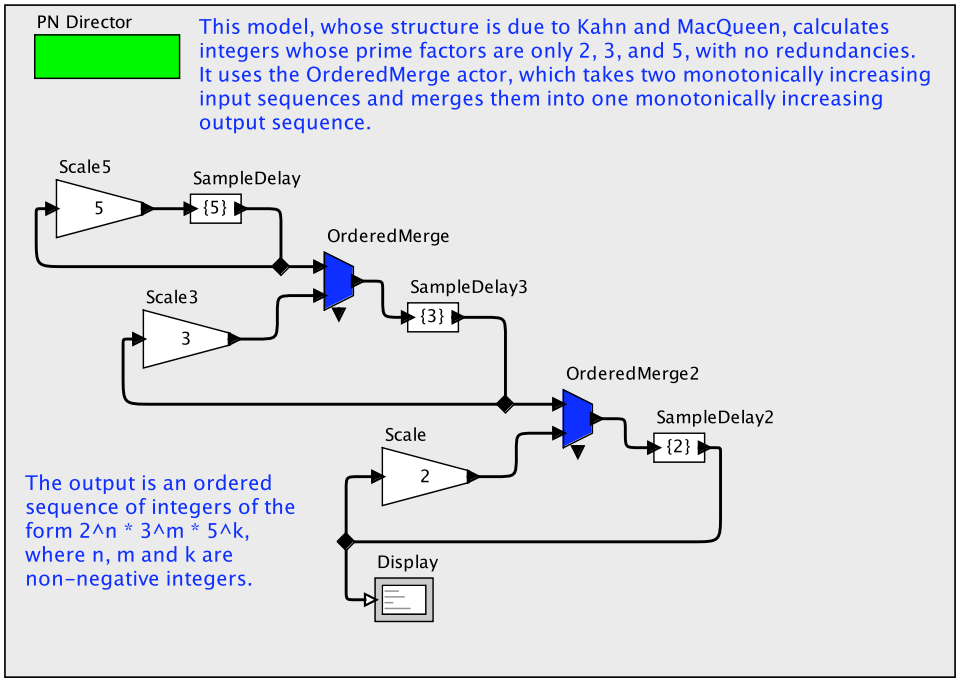


Figure 1: Model that generates a sequence of Hamming numbers.