Homework No. 1

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, solve problems 1.5 and 1.10. These are exercises with ordered sets, the prefix order, and order-preserving functions. They are reproduced below for convenience:

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.

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

2 Design Exercises

The purpose of these exercises is to develop some intuition about the process networks model of computation and how programming with it differs from programming with an imperative model. You will need an up-to-date installation of Ptolemy II, from the SVN repository. You may want to run vergil with the -pn option, which gives you a subset of Ptolemy II that is more than adequate to do this homework. To do this on the command line, simply type

```
>> vergil -pn
```

If you are running Ptolemy II from Eclipse, then in the toolbar of the Java perspective, select Run Configurations. In the Arguments tab, enter -pn.

For all of the following exercises, you should use "simple" actors to accomplish the task. In particular, the following actors are sufficient:

- Ramp and Const (in Actors: Sources)
- Display and Discard (in Actors: Sinks)
- BooleanSwitch and BooleanSelect (in Actors: FlowControl: BooleanFlowControl)
- SampleDelay (in Actors: FlowControl: SequenceControl)

- Comparator, Equals, LogicalNot, or LogicFunction (in Actors: Logic)

Feel free to use any other actor that you believe to be "simple" (you may come up with better solutions than mine). Also, feel free to use any other actors, simple or not, for testing your composite actors, but stick to simple ones for the implementation of the composite actors.

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.

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.

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.

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.

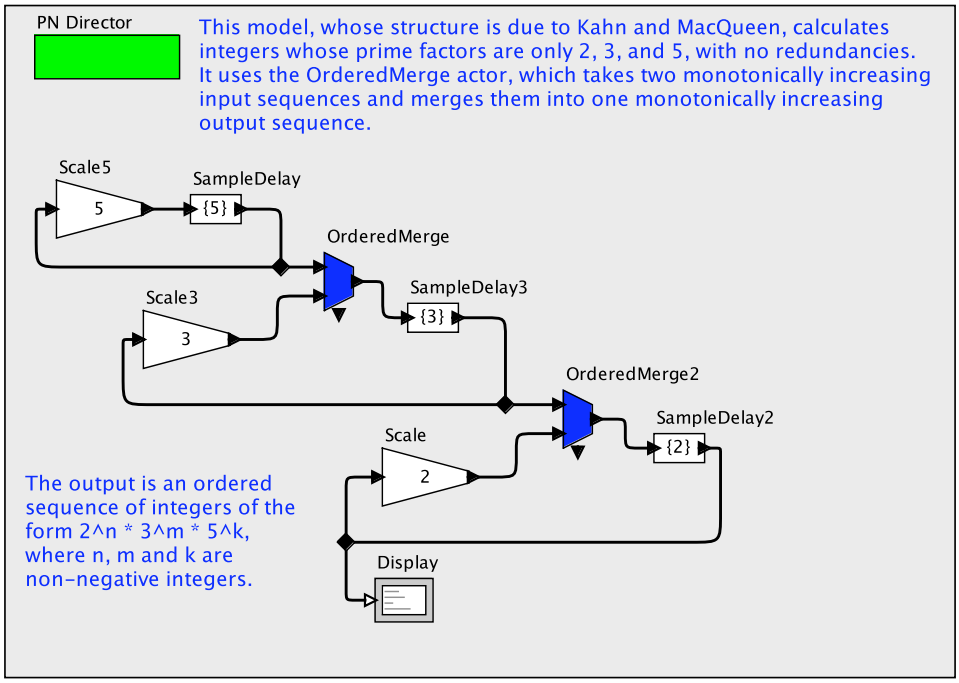

Figure 1: Model that generates a sequence of Hamming numbers.