Homework 4

EE 290n - Advanced Topics in Systems Theory

Edward A. Lee

1. Suppose D is some set and $S = D^{**}$ is the set of finite and infinite sequences of elements of D. This exercise explores some of the properties of the CPO S^n with the pointwise prefix order, for some non-negative integer n. In particular, these properties are useful for understanding dataflow process networks.

   (a) Show that any two elements $a, b \in S^n$ that have an upper bound have a least upper bound.

   (b) Let $U \subset S^n$ be such that no two distinct elements of U are joinable. Prove that for all $s \in S^n$ there is at most one $u \in U$ such that $u \sqsubseteq s$.

   (c) Given $s \in S^n$, suppose that $Q(s) \subset S^n$ is a joinable set where for all $q \in Q(s)$, $q \sqsubseteq s$. Then show that there is an s' such that $s = (\bigvee Q(s)).s'$.

   Solution.

   (a) Let c be an upper bound of a and b. The $a \sqsubseteq c$ and $b \sqsubseteq c$. Under the pointwise prefix order, this implies that $\pi_i(a) \sqsubseteq \pi_i(c)$ and $\pi_i(b) \sqsubseteq \pi_i(c)$ for each $i \in \{1, \cdots, n\}$. Since $\pi_i(a)$ and $\pi_i(b)$ are ordinary sequences, if they are both prefixes of the same sequence $\pi_i(c)$, then it must be that either $\pi_i(a) \sqsubseteq \pi_i(b)$ or $\pi_i(b) \sqsubseteq \pi_i(a)$. We can construct a $d \in S^n$ where $\pi_i(d)$ is defined to be $\pi_i(b)$ if $\pi_i(a) \sqsubseteq \pi_i(b)$, and is defined to be $\pi_i(a)$ otherwise, for each $i \in \{1, \cdots, n\}$. Then clearly d is an upper bound of a and b, and moreover, $\pi_i(d) \sqsubseteq \pi_i(c)$ for each $i \in \{1, \cdots, n\}$, so d is a least upper bound under the pointwise prefix order.

   (b) Note first that the theorem is trivially true for $n = 0$. For $n > 0$, assume to the contrary that you have two distinct $u, u' \in U$ such that $u \sqsubseteq s$ and $u' \sqsubseteq s$ for some $s \in S^n$. Then s is an upper bound for $\{u, u'\}$. From part (a), $\{u, u'\}$ has a least upper bound, and hence u and u' are joinable, contradicting the assumption that no two distinct elements of U are joinable.

   (c) It is sufficient to show that $\bigvee Q(s) \sqsubseteq s$. Note first this is trivially true for $n = 0$, so we henceforth assume $n > 0$. Consider each dimension $i \in \{1, \cdots, n\}$. For each such i, there is a $q \in Q(s)$ such that $\pi_i(\bigvee Q(s)) = \pi_i(q)$. We know that $\pi_i(q) \sqsubseteq \pi_i(s)$, so we conclude that $\pi_i(\bigvee Q(s)) \sqsubseteq \pi_i(s)$ for each such i. Hence, $\bigvee Q(s) \sqsubseteq s$.

   □

2. Consider the model shown in figure 1. Assume that data types are all $D = \{0, 1\}$. Assume f is a dataflow actor that implements an identity function and that Const is an actor that produces an infinite sequence $(0, 0, 0, \cdots)$. Obviously, the overall output of this model should be this same infinite sequence. The box labeled g indicates a composite actor. Find firing rules and firing function g for the composite actor to satisfy conditions 1 and 3 covered in class. Note that the composite actor has one input and two outputs.

Solution. Let $U = \{(0),(1),\perp\}$ be the set of firing rules. Note that subsets $\{(0),\perp\}$ and $\{(1),\perp\}$ are joinable. Notice that the greatest lower bound of each of these sets is $\perp$, so the first part of rule 3 is satisfied. Let g be defined so that

$$\begin{aligned} g((0)) &= ((0),\perp) \qquad (1)\\ g((1)) &= ((1),\perp) \qquad (2)\\ g(\perp) &= (\perp,(0)). \qquad (3)\end{aligned}$$

Note that this firing function yields, as desired, and infinite sequence $(0,0,0,\cdots)$. Note now that if $u = (0)$ and $u' = \perp$, then

$$g(u).g(u') = g(u').g(u).$$

The same is true if $u = (1)$ and $u' = \perp$, so the rest of rule 3 is satisfied. □

3. The Dynamic Dataflow director in Ptolemy II supports an actor called ActorRecursion, created by Gang Zhou. It can be used in model as a recursive reference to a composite actor that contains it. For example, the Prime_Number_Filter demo, shown in figure 2, implements the sieve of Eratosthenes, as described in the Kahn and MacQueen paper.

   Use this actor to implement a composite actor that computes Fibonacci numbers. That is, a firing of your composite actor should implement the firing function $f\colon \mathbb{N} \to \mathbb{N}$ defined by

$$f(n) = \begin{cases} 0 & \text{if } n = 0\\ 1 & \text{if } n = 1\\ f(n-1)+f(n-2) & \text{otherwise}\end{cases}$$

   for all $n \in \mathbb{N}$.

   The way that this actor works is that when it fires, it clones the composite actor above it in the hierarchy (i.e., its container, or its container's container, etc.) whose name matches the value of its *recursionActor* parameter. The instance of ActorRecursion is populated with ports that match those of that container.

   This actor should be viewed as a highly experimental realization of a particular kind of higher-order component. It is a higher-order component because it is parameterized by an actor that

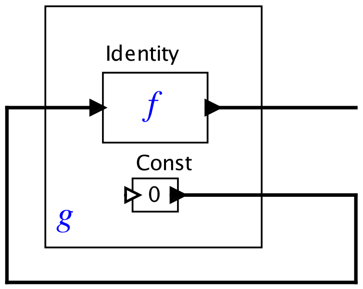


Figure 1: A model.

contains it. Its implementation has a number of issues, each of which could be interesting research directions:

(a) It is very inefficient. The cloning of the actor it references on each firing is expensive in both memory and time. A better implementation would use something like the stack frame approach used in procedural programming languages. Instead, the approach it uses is more like copying the source code at run time and then interpreting it. In an attempt to make execution more efficient, this actor avoids creating the clone if it has previously created it. However, this leads to the second problem.

(b) If you run a model with an ActorRecursion instance in it, upon completion of the run, the actor will contain the clone. You can look inside the actor to see this. However, if you now make a modification to the referenced actor, the clone is not recreated on the next run, so your modifications will not be reflected. Apparently, you have to save your model, exit Ptolemy II, and then re-enter to re-run the model. This is a bug, probably easily fixed, but I mention it to help avoid frustration in using this actor.

(c) The actor is written to work with the Dynamic Dataflow director, but this constraint is probably not really necessary. The same mechanism would probably work with several other directors, including at least PN and DE. An interesting question is whether there is some variant that would work with SR. To my knowledge, none of the synchronous languages support such recursion.

(d) The visual representation of the recursive reference is inadequate. There is no way, looking only at the image in figure 2, to tell what composite actor the ActorRecursion

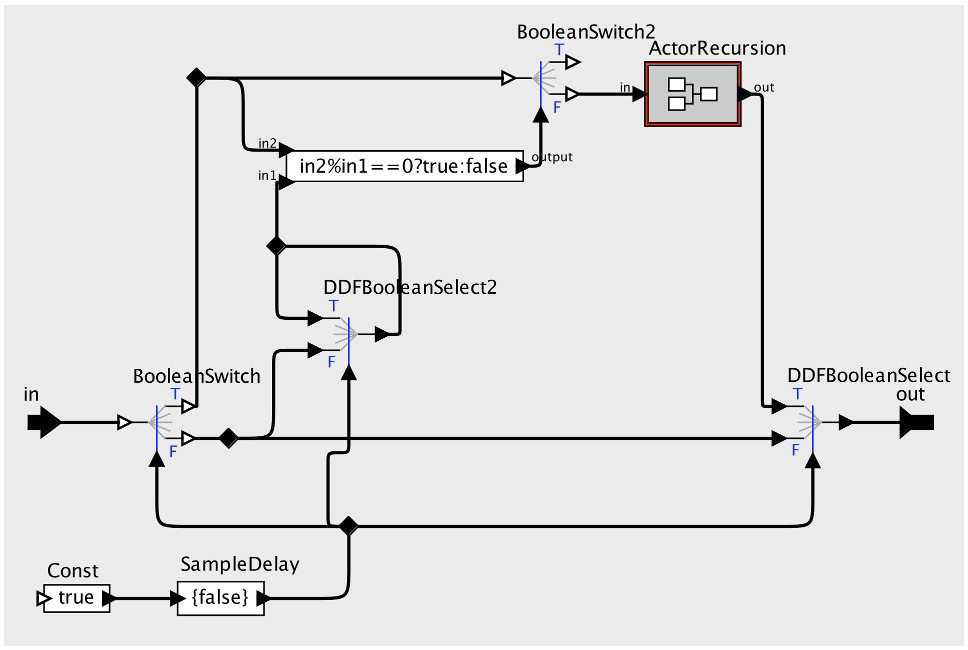
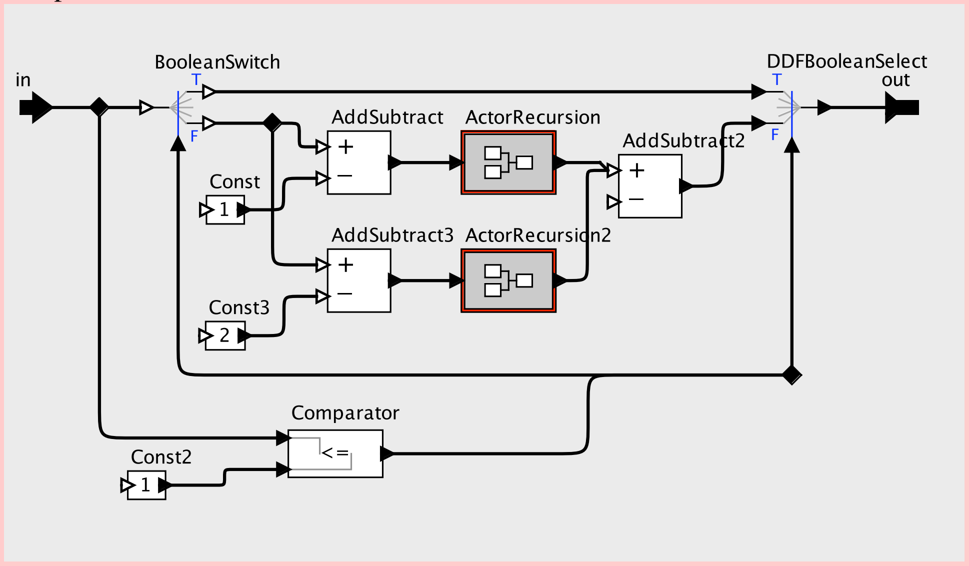


Figure 2: The sieve of Eratosthenes, using ActorRecursion in Dynamic Dataflow, created by Gang Zhou.

instance references. Thus, you cannot really read the program from its printout.

Ideas and fixes to any of the above issues are welcome.

Solution. A solution is shown below, where each ActorRecursion actor references the composite shown below:



□