



Concurrent Models of Computation for Embedded Software

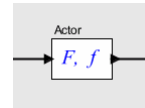
Edward A. Lee

Professor, UC Berkeley
EECS 290n – Advanced Topics in Systems Theory
Fall, 2004

Copyright © 2004, Edward A. Lee, All rights reserved

Lecture 15: Generalized Firing Rules

Firing Rules from Last Lecture



Let $F : S^n \rightarrow S^m$ be a dataflow process.

Let $U \subset S^n$ be a set of *firing rules* with the constraints:

1. Every $u \in U$ is finite, and
2. No two elements of U are joinable.

This implies that for all $s \in S^n$ there is at most one $u \in U$ where $u \sqsubseteq s$. (exercise)

When $u \sqsubseteq s$ there is a unique s' such that $s = u.s'$ where the period denotes concatenation of sequences.

Source and Sink Actors

Sink actor: $F : S^n \rightarrow S^0$ with firing function $f : S^n \rightarrow S^0$.

In this case, if $S^0 = \{\sigma\}$ then $f(u) = \sigma$ is the single element. Define concatenation in S^0 so that $\sigma.\sigma = \sigma$. Then everything works (e.g., let $\sigma = \perp$).

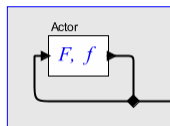
Source actor: $F : S^0 \rightarrow S^m$ with firing function $f : S^0 \rightarrow S^m$. Firing rules $U = S^0$ (singleton set) have the constraints trivially satisfied.

Lee 15: 3

Source Actors Too Limited?

With the above definitions, the dataflow process produces the sequence $f(\sigma).f(\sigma).f(\sigma) \dots$ where $U = S^0 = \{\sigma\}$.

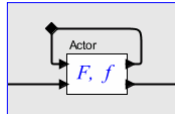
If U is non-empty, this is infinite and periodic. This may seem limiting for dataflow processes that act as sources, but in fact it is not, because a source with a more complicated output sequence can be constructed using feedback composition.



Lee 15: 4

More Generally: Is a Single Firing Function Too Restrictive?

Not really. Use a self loop:



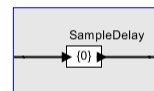
Let the data type of the feedback loop be $V = \{1, 2, \dots, n\}$

Then the first argument to the firing function can represent n different “states” of the actor, where in each state the output is a different function of the input.

But how can you get this started?

Lee 15: 5

A Possible Problem: Sample Delay Actor

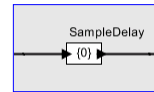


Can the sample delay be represented with the following firing rules?

$$\{\perp, (0), (1)\}$$

Lee 15: 6

A Possible Problem: Sample Delay Actor



Can the sample delay be represented with the following firing rules?

$$\{\perp, (0), (1)\}$$

No. These are not joinable.

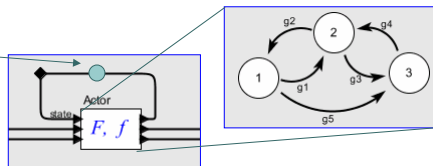
Instead, we require that initial tokens on an arc be a primitive concept in dataflow. This is implemented in Ptolemy II by outputting the initial token prior to any firings.

Lee 15: 7

Firing Rules Defined by a State Machine

Feedback path data type: $V = \{1, 2, \dots, n\}$ where there are n states:

initial state $i \in V$



In each state $i \in V$, there is a set of firing rules

$$U_i = \{(i, \dots), (i, \dots), \dots\}$$

where every member is finite and no two members are joinable. Then the total set of firing rules is

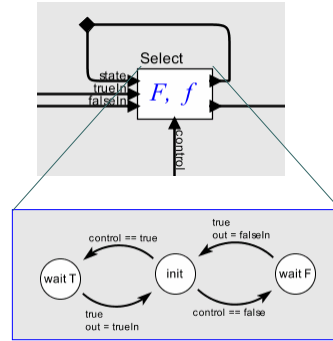
$$U = U_1 \cup \dots \cup U_n$$

Every member is finite and no two members are joinable.

Lee 15: 8

Example: Select Actor

- In the *init* state, read input from the *control* port.
- In the *waitT* state, read input from the *trueIn* port.
- In the *waitF* state, read input from the *falseIn* port.



$$U_{init} = \{(init, \perp, \perp, *)\}$$

$$U_{waitT} = \{(waitT, *, \perp, \perp)\}$$

$$U_{waitF} = \{(waitF, \perp, *, \perp)\}$$

shorthand to match any input token

Lee 15: 9

Sequential Functions

Any sequential function can be implemented by a state machine that in each state has firing rules that match the state identifier in the state input port and match any token in exactly one other input port.

Each state could also (in effect) implement a different firing function (one firing function with the state identifier as an input can model this).

Lee 15: 10

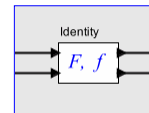
Generalize Further to get the Cal Actor Language

Partition the firing rules and associate a distinct firing function with each partition of the firing rules. Each such firing function is called an *action*.

This is similar to the pattern matching in some functional languages such as Haskell.

Lee 15: 11

Another Possible Problem: Cannot Implement Identity Functions!



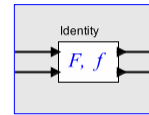
Will the following firing rules work?

$$\{((0), \perp), ((1), \perp), (\perp, (0)), (\perp, (1))\}$$

$$\{((0), (0)), ((0), (1)), ((1), (0)), ((1), (1))\}$$

Lee 15: 12

Cannot Implement Identity Functions!



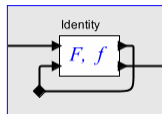
Will the following firing rules work?

$\{((0), \perp), ((1), \perp), (\perp, (0)), (\perp, (1))\}$

No. Nondeterminate merge.

$\{((0), (0)), ((0), (1)), ((1), (0)), ((1), (1))\}$

No. Try feeding back one output to one input. E.g.:



Lee 15: 13

Generalized Firing Rules

We previously defined the firing rules $U \subset S^n$ with:

1. Every $u \in U$ is finite, and
2. No two elements of U are joinable.

We now replace constraint 2 with:

3. For any two elements of $u, u' \in U$ that are joinable, we require that:

$$u \wedge u' = \perp_n$$
$$f(u) \cdot f(u') = f(u') \cdot f(u)$$

I.e., when two firing rules are enabled, they can be applied in either order without changing the output.

Lee 15: 14

Examining Rule 3

3. For any two elements of $u, u' \in U$ that are joinable, we require that:

$$u \wedge u' = \perp_n$$

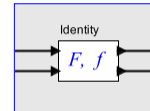
I.e., no two joinable firing rules have a common prefix.

$$f(u) \cdot f(u') = f(u') \cdot f(u)$$

I.e., when two firing rules are enabled, they can be applied in either order without changing the output.

Lee 15: 15

Applying Rule 3 to Identity Functions



With these firing rules

$$U = \{((0), \perp), ((1), \perp), (\perp, (0)), (\perp, (1))\}$$

and for all $u \in U$,

$$f(u) = u$$

rule 3 is satisfied. Exercise: Show that rule 3 is not satisfied by the nondeterminate merge.

Lee 15: 16

Fixed Point Semantics Under Rule 3

Let $Q(s) = \{u_1, u_2, \dots, u_q\} \subset U$ be the set of all firing rules that are a prefix of s . This could be empty. Then define

$$(\phi'(F))(s) = \begin{cases} f(u_1).f(u_2).\dots.f(u_q).F(s') & \text{if } Q(s) \neq \emptyset \\ \perp_n & \text{otherwise} \end{cases}$$

Where $s = \vee Q(s).s'$
(exercise to show that s' always exists).

The function ϕ' is continuous, and all previous results hold.

Lee 15: 17

Conclusions and Open Issues

- Dataflow processes are Kahn processes composed of atomic *firings*.
- Firing rules that are not joinable lead to simple fixed point semantics.
- Simple semantics leaves out delays, two-input identity functions, and other compositions.
- Generalized firing rules allow joinable pairs under certain circumstances.

Lee 15: 18