Concurrent Models of Computation

Edward A. Lee
Robert S. Pepper Distinguished Professor, UC Berkeley

EECS 219D
Concurrent Models of Computation
Fall 2011

Week 8: Dataflow Process Networks

Firings

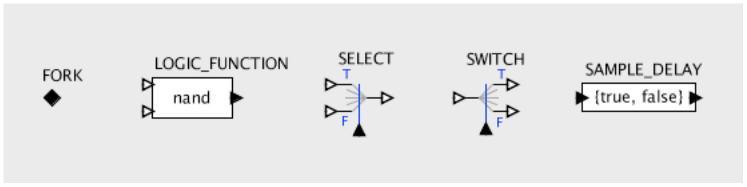
Dataflow is a variant of Kahn Process Networks where a process is computed as a sequence of atomic *firings*, which are finite computations enabled by a *firing rule*.

In a firing, an actor consumes a finite number of input tokens and produces a finite number of outputs.

A possibly infinite sequence of firings is called a *dataflow process.*

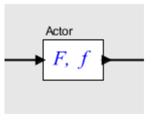
PN actors as sequences of firings

The following actors can be described denotationally as functions over sequences of input values, or operationally as sequences of finite computations called "firings." Each firing consumes a finite amount of input data and produces a finite amount of output data.



When a PN process can be described this way, it is called a *dataflow process.*

Firing Rules



Let $F : S^n \to S^m$ be a dataflow process, where $S = D^{**}$.

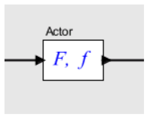
Let $U \subset S^n$ be a set of *firing rules* with the constraints:

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.

Firing Function



Let $f : S^n \to S^m$ be a (possibly partial) firing function with the constraint that for all $u \in U$, $f(u)$ is defined and is finite.

Then the dataflow process $F : S^n \to S^m$ is given by

$$F(s) = \begin{cases} f(u).F(s') & \text{if there is a } u \in U \text{ such that } s = u.s' \\ \perp_n & \text{otherwise} \end{cases}$$

where $\perp_n \in S^n$ is the n-tuple of empty sequences.
Note that this is self referential. Seek a fixed point F.

Fixed Point Definition of Dataflow Process (cf. Lifting Formulation in SR)

Define $\phi : [S^n \to S^m] \to [S^n \to S^m]$ by:

$$(\phi(F))(s) = \begin{cases} f(u).F(s') & \text{if there is a } u \in U \text{ such that } s = u.s' \\ \perp_n & \text{otherwise} \end{cases}$$

Fact: ϕ is continuous (see Lee & Matsikoudis). This means that it has a unique least fixed point, and that we can constructively find that fixed point by starting with the bottom of the CPO. The bottom of the CPO is the function $F_0 : S^n \to S^m$ that returns $\perp_n$.

Executing a Dataflow Process is the Same as Finding the Least Fixed Point

Suppose $s \in S^n$ is a concatenation of firing rules,

$$s = u_1 . u_2 . u_3 \quad \ldots$$

Then the procedure for finding the least fixed point of ϕ yields the following sequence of approximations to the dataflow process:

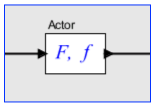
$$F_0(s) = \bot_n$$
$$F_1(s) = (\phi(F_0))(s) = f(u_1)$$
$$F_2(s) = (\phi(F_1))(s) = f(u_1).f(u_2)$$
$$\ldots$$

This exactly describes the operational semantics of repeated firings governed by the firing rules!

The LUB of this Sequence of Functions is Continuous

The chain $\{F_0(s), F_1(s), \ldots\}$ will be finite for some s (certainly for finite s, but also for any s for which after some point, no more firing rules match), and infinite for other s. Since each F_i is a continuous function, and the set of continuous functions is a CPO, then the LUB is continuous, and hence describes a valid Kahn process that guarantees determinacy, and can be put into a feedback loop.

Example 1



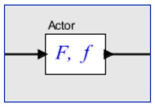
Suppose $D = \{0, 1\}$ and $S = D^{**}$ is the set of finite and infinite sequences of elements from D .

Consider a dataflow process with one input and one output, $F : S \rightarrow S$. Its firing rules are $U \subset S$. The following are all valid firing rules:

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

Lee 08: 9

Example 2 : Valid Firing Rule?



Suppose $D = \{0, 1\}$ and $S = D^{**}$ is the set of finite and infinite sequences of elements from D .

Consider a dataflow process with one input and one output, $F : S \rightarrow S$. Its firing rules are $U \subset S$. Is the following set a valid set of firing rule?

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

Lee 08: 10

Example 2 : Valid Firing Rule?

Suppose $D = \{0, 1\}$ and $S = D^{**}$ is the set of finite and infinite sequences of elements from D .

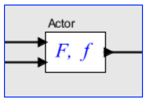
Consider a dataflow process with one input and one output, $F : S \rightarrow S$. Its firing rules are $U \subset S$. Is the following set a valid set of firing rule?

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

No. There are joinable pairs.

Intuition: The same input sequence can lead to multiple executions. Nondeterminacy!

Example 3



Consider $F : S^2 \rightarrow S$. Its firing rules are $U \subset S^2$. Which of the following are valid sets of firing rules?

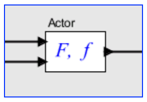
$$\{((0), (0)), ((0), (1)), ((1), (0)), ((1), (1))\}$$

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

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

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

Example 3



Consider $F : S^2 \to S$. Its firing rules are $U \subset S^2$. Which of the following are valid sets of firing rules?

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

Yes. Consume one token from each input.

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

No. Nondeterminate merge.

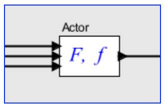
$$\{((0), \perp), ((1), (0)), ((1), (1))\}$$

Yes. Consume from the second input if the first is 1.

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

Yes. Consume only from the first input.

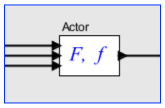
Example 4



Consider $F : S^3 \to S$. Its firing rules are $U \subset S^3$. Is the following a valid set of firing rules?

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

Example 4



Consider $F : S^3 \to S$. Its firing rules are $U \subset S^3$. Is the following a valid set of firing rules?

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

Yes. Dataflow version of the Gustave function!

Taking Stock

- Dataflow processes are Kahn processes composed of atomic *firings*.
- Firing rules that are not joinable lead to simple fixed point semantics.

Source and Sink Actors

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

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: $F : S^0 \to S^m$ with firing function $f : S^0 \to S^m$.

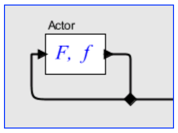
Firing rules $U = S^0$ (singleton set) have the constraints trivially satisfied.

Lee 08: 17

Are Source Actors Too Limited?

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

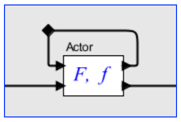
If is non-empty, this is infinite and periodic. This may seem limiting for dataflow processes that act as sources, but in fact it is not, because a source with a more complicated output sequence can be constructed using feedback composition.



Lee 08: 18

More Generally: Is a Single Firing Function Too Restrictive?

Not really. Use a self loop:

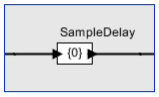


Let the data type of the feedback loop be $V = \{1, 2, \ldots , n\}$

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?

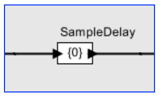
A Possible Problem: Sample Delay Actor



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

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

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.

One option: require that initial tokens on an arc be a primitive concept in dataflow.

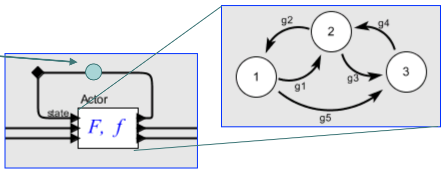
(An alternative is to make state machines a primitive concept).

Lee 08: 21

Firing Rules Defined by a State Machine

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

initial state $i \in V$



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

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

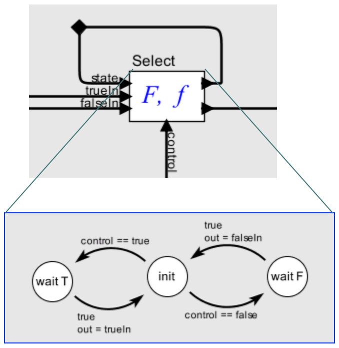
where every member is finite and no two members are joinable. Then the total set of firing rules is

$$U = U_1 \cup \ldots \cup U_n$$

Every member is finite and no two members are joinable.

Lee 08: 22

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 08: 23

Recall *sequential* Functions [Vuillemin]

Let $f: A^n \to A^m$ be an n input, m output function.

Then f is *sequential* if it is continuous and for any $a \in A^n$ there exists an $i \in \{1, \dots n\}$, such that for all $b \in A^n$ where $a \le b$,

$$a|_{\{i\}} = b|_{\{i\}} \Rightarrow f(a) = f(b)$$

Intuitively: At all times during an execution, there is an input channel that blocks further output. This is the Kahn-MacQueen blocking read!

Lee 08: 24

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.

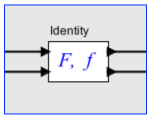
Each state could also (in effect) implement a different firing function (one firing function with the state identifier as an input can model this).

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.

Another Possible Problem: Cannot Implement Identity Functions!



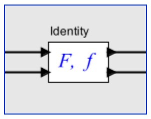
Will the following firing rules work?

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

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

Lee 08: 27

Cannot Implement Identity Functions!



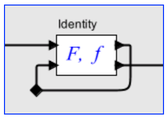
Will the following firing rules work?

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

No. Nondeterminate merge.

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

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



Lee 08: 28

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:

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

$$f(u) \,.\, f(u') = f(u') \,.\, f(u)$$

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

Examining Rule 3

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

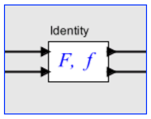
$$u \wedge u' = \perp_n$$

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

$$f(u) \,.\, f(u') = f(u') \,.\, f(u)$$

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

Applying Rule 3 to Identity Functions



With these firing rules

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

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.

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

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

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

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

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.