



Concurrent Models of Computation for Embedded Software

Edward A. Lee

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

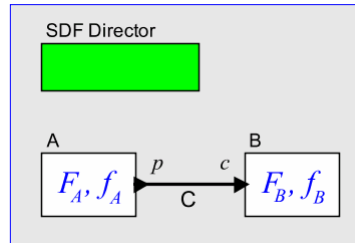
Copyright © 2004, Edward A. Lee, All rights reserved

Lecture 18: Boolean Dataflow

History of Dataflow Models of Computation

- o Computation graphs [Karp & Miller - 1966]
- o Process networks [Kahn - 1974]
- o Static dataflow [Dennis - 1974]
- o Dynamic dataflow [Arvind, 1981]
- o K-bounded loops [Culler, 1986]
- o Synchronous dataflow [Lee & Messerschmitt, 1986]
- o Structured dataflow [Kodosky, 1986]
- o PGM: Processing Graph Method [Kaplan, 1987]
- o Synchronous languages [Lustre, Signal, 1980's]
- o Well-behaved dataflow [Gao, 1992]
- o Boolean dataflow [Buck and Lee, 1993] today
- o Multidimensional SDF [Lee, 1993]
- o Cyclo-static dataflow [Lauwereins, 1994]
- o Integer dataflow [Buck, 1994]
- o Bounded dynamic dataflow [Lee and Parks, 1995]
- o Heterochronous dataflow [Girault, Lee, & Lee, 1997]
- o Parameterized dataflow [Bhattacharya and Bhattacharyya 2001] Friday
- o ...

Statically Schedulable Dataflow – SSDF Historically called: Synchronous Dataflow (SDF)



If the number of tokens consumed and produced by the firing of an actor is constant, then static analysis can tell us whether we can schedule the firings to get a useful execution, and if so, then a finite representation of a schedule for such an execution can be created.

Lee 18: 3

Expressiveness Limitations in SSDF

SSDF cannot express data-dependent flow of tokens:

- If-then-else
- Do-while
- Recursion

Hierarchical SSDF can do some of this...

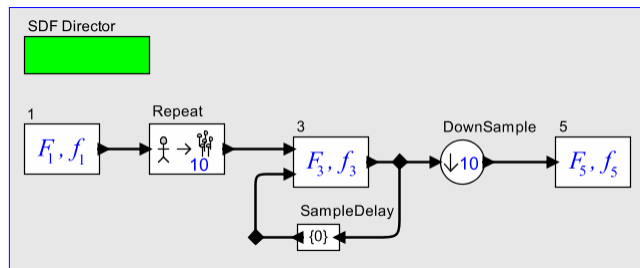
A more general solution is dynamically scheduled dataflow. We now explore DDF, and in particular, how to use static analysis to achieve similar results to those of SSDF.

Lee 18: 4

Manifest Iteration in SSDF

Imperative equivalent:

```
while (true) {
  x = f1();
  y = 0;
  for I in (1..10) {
    y = f3(x, y);
  }
  f5(y);
}
```



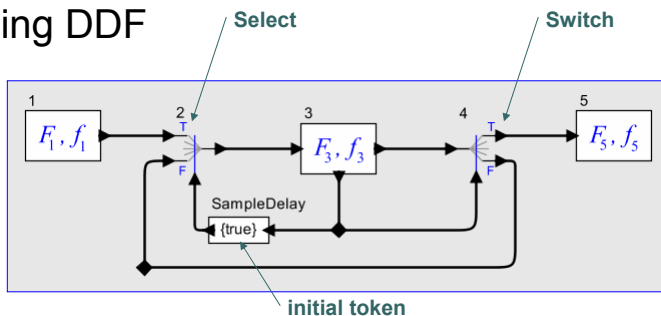
Manifest iteration (where the number of iterations is a fixed constant) is expressible in SSDF. But data-dependent iteration is not.

Lee 18: 5

Do-While Using DDF

Imperative equivalent:

```
while (true) {
  x = f1();
  b = false;
  while(!b) {
    (x, b) = f3(x);
  }
  f5(x);
}
```

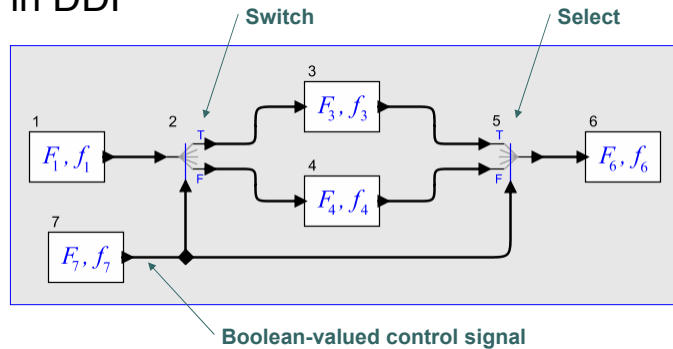


This model uses conditional routing of tokens to iterate a function a data-dependent number of times.

Exercise: Can this be done with HDF? Hierarchical SSDF?

Lee 18: 6

If-Then-Else in DDF



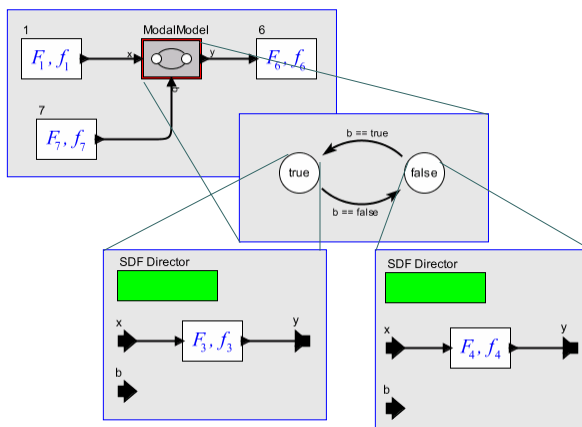
Imperative equivalent:

```
while (true) {
    x = f1();
    b = f7();
    if (b) {
        y = f3(x);
    } else {
        y = f4(x);
    }
    f6(y);
}
```

This model uses conditional routing of tokens to route each token in a stream through one of two actors.

Lee 18: 7

Aside: Compare With If-Then-Else Using Heterochronous Dataflow



Imperative equivalent:

```
b = true;
while (true) {
    x = f1();
    if (b) {
        y = f3(x);
    } else {
        y = f4(x);
    }
    f6(y);
    b = f7();
}
```

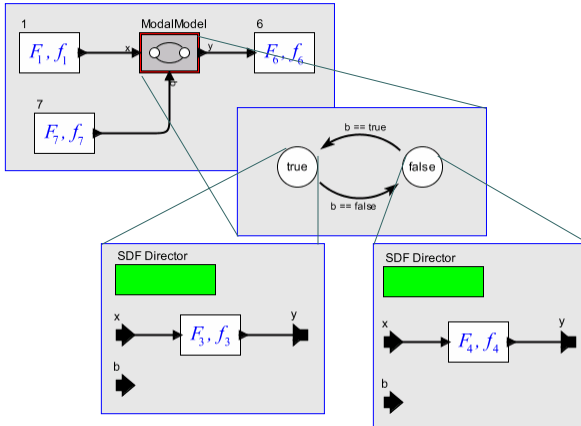
Note that this is not quite the same as the previous version...

Semantics of HDF:

- Execute SDF model for one complete iteration in current state
- Take state transitions to get a new SDF model.

Lee 18: 8

Aside: Compare With If-Then-Else Using Heterochronous Dataflow



Imperative equivalent:

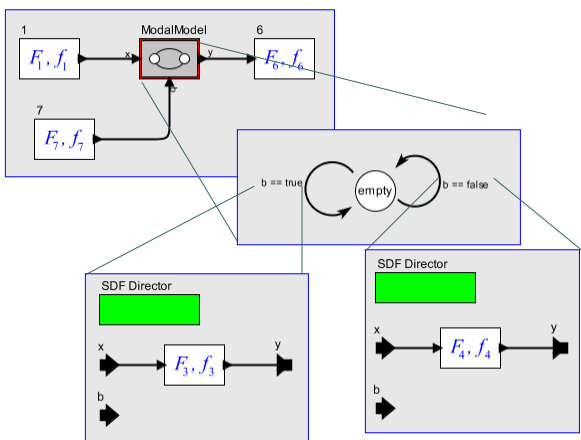
```

b = true;
while (true) {
    x = f1();
    if (b) {
        y = f3(x);
    } else {
        y = f4(x);
    }
    f6(y);
    b = f7();
}
    
```

Note that if these two refinements have the same production/consumption parameters, then this is simply hierarchical SDF, where one static schedule suffices.

Lee 18: 9

Hierarchical SDF Using Transition Refinements



Imperative equivalent:

```

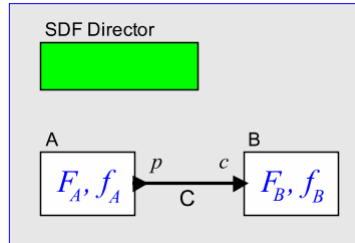
while (true) {
    x = f1();
    b = f7();
    if (b) {
        y = f3(x);
    } else {
        y = f4(x);
    }
    f6(y);
}
    
```

This only works under rather narrow constraints:

- Exactly one outgoing transition from any state is enabled.
- The transition refinements on all transitions have the same production/consumption patterns.
- The state has no refinement.

Lee 18: 10

Balance Equations



Let q_A, q_B be the number of firings of actors A and B.

Let p_C, c_C be the number of token produced and consumed on a connection C.

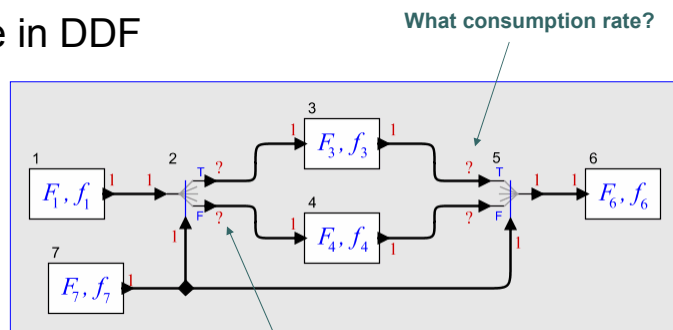
Then the system is *in balance* if for all connections C

$$q_A p_C = q_B c_C$$

where A produces tokens on C and B consumes them.

Lee 18: 11

If-Then-Else in DDF



Imperative equivalent:

```
while (true) {
  x = f1();
  b = f7();
  if (b) {
    y = f3(x);
  } else {
    y = f4(x);
  }
  f6(y);
}
```

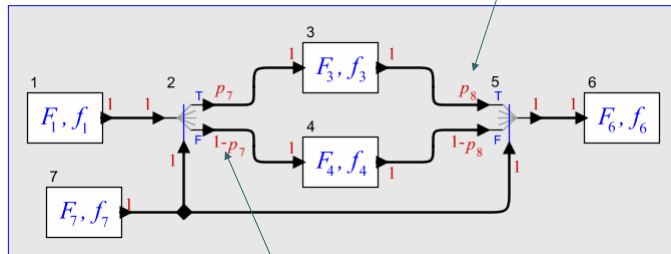
The if-then-else model is not SDF. But we can clearly give a bounded *quasi-static* schedule for it:
 (1, 7, 2, b?3, !b?4, 5, 6)

guard

Lee 18: 12

Symbolic Rates

Symbolic consumption rate.



Imperative equivalent:

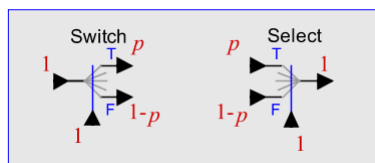
```
while (true) {
  x = f1();
  b = f7();
  if (b) {
    y = f3(x);
  } else {
    y = f4(x);
  }
  f6(y);
}
```

Production and consumption rates are given symbolically in terms of the values of the Boolean control signals consumed at the control port.

Symbolic production rate.

Lee 18: 13

Interpretations of Symbolic Rates

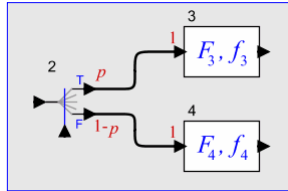


- General interpretation: p is a symbolic placeholder for an unknown.
- Probabilistic interpretation: p is the probability that a Boolean control input is *true*.
- Proportion interpretation: p is the proportion of *true* values at the control input in one *complete cycle*.

NOTE: We do not need numeric values for p . We always manipulate it symbolically.

Lee 18: 14

Symbolic Balance Equations



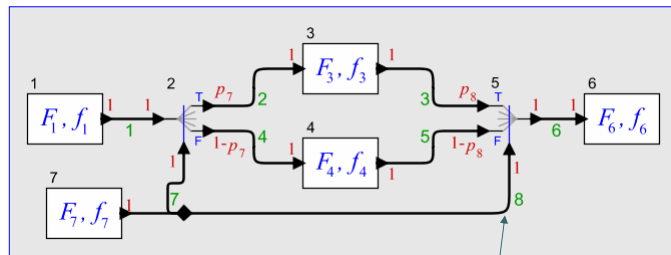
The two connections above imply the following balance equations:

$$q_2 p = q_3$$

$$q_2 (1 - p) = q_4$$

Lee 18: 15

Symbolic Rates



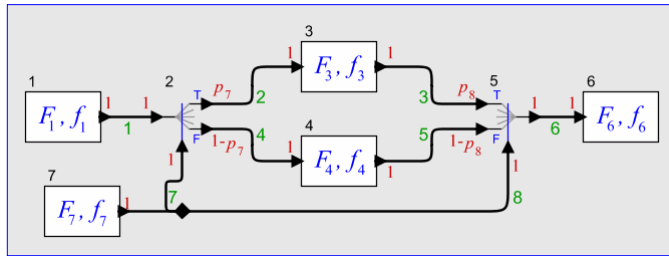
Imperative equivalent:

```
while (true) {
  x = f1();
  b = f7();
  if (b) {
    y = f3(x);
  } else {
    y = f4(x);
  }
  f6(y);
}
```

Production and consumption rates are given symbolically in terms of the values of the Boolean control signals consumed at the control port.

Lee 18: 16

Production/Consumption Matrix for If-Then-Else



Symbolic variables:

$$\vec{p} = \begin{bmatrix} p_7 \\ p_8 \end{bmatrix}$$

$$\Gamma(\vec{p}) = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & p_7 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -p_8 & 0 & 0 \\ 0 & 1-p_7 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -(1-p_8) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

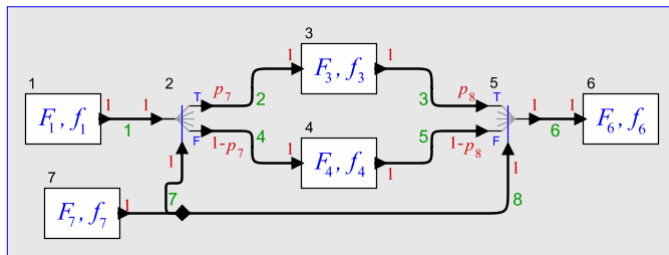
Balance equations:

$$\Gamma(\vec{p})q(\vec{p}) = \vec{0}$$

Note that the solution $q(\vec{p})$ now depends on the symbolic variables \vec{p}

Lee 18: 17

Production/Consumption Matrix for If-Then-Else



The balance equations have a solution $q(\vec{p})$ if and only if $\Gamma(\vec{p})$ has rank 6. This occurs if and only if $p_7 = p_8$, which happens to be true by construction because signals 7 and 8 come from the same source. The solution is given at the right.

$$q(\vec{p}) = \begin{bmatrix} 1 \\ 1 \\ p_7 \\ 1-p_7 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Lee 18: 18

Strong and Weak Consistency

A *strongly consistent* dataflow model is one where the balance equations have a solution that is provably valid without concern for the values of the symbolic variables.

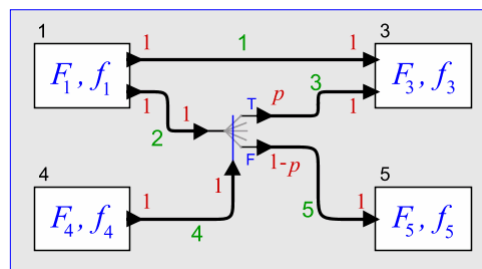
- The if-then-else dataflow model is strongly consistent.

A *weakly consistent* dataflow model is one where the balance equations cannot be proved to have a solution without constraints on the symbolic variables that cannot be proved.

- Note that whether a model is strongly or weakly consistent depends on how much you know about the model.

Lee 18: 19

Weakly Consistent Model



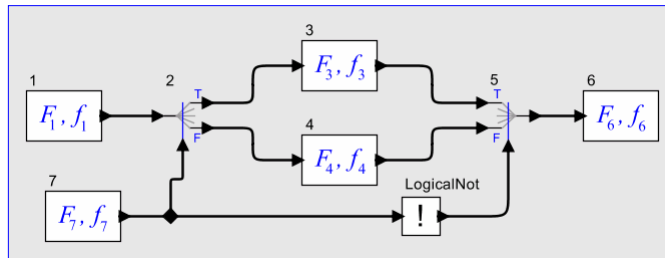
$$\Gamma(\vec{p}) = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & p & -1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 1-p & 0 & 0 & -1 \end{bmatrix}$$

This production/consumption matrix has full rank unless $p = 1$.

Unless we know f_4 , this cannot be verified at compile time.

Lee 18: 20

Another Example of a Weakly Consistent Model

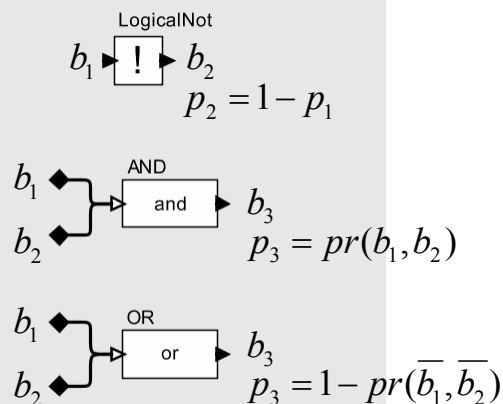


This one requires that actor 7 produce half true and half false (that $p = 0.5$) to be consistent. This fact is derived automatically from solving the balance equations.

Lee 18: 21

Use Boolean Relations

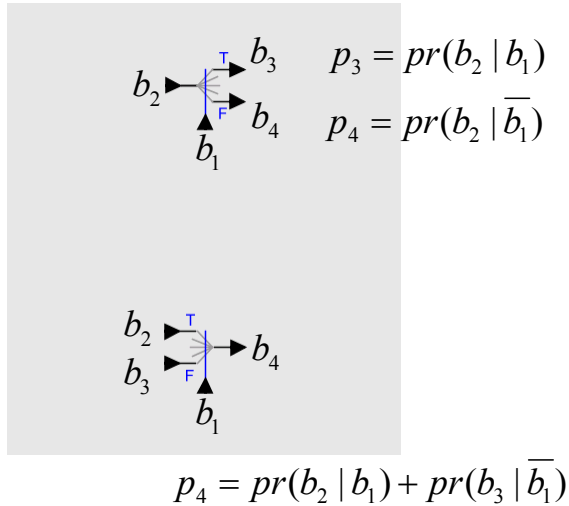
Symbolic variables across logical operators can be related as shown.



Lee 18: 22

Routing of Boolean Tokens

Symbolic variables across switch and select can be related as shown.



Lee 18: 23

Conclusions and Open Issues

- BDF generalizes the idea of balance equations to include symbolic variables.
- Whether balance equations have a solution may depend on the relationships between symbolic variables.

Lee 18: 24