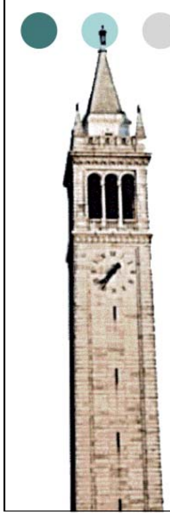
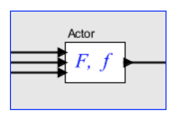


Concurrent Models of Computation

Edward A. Lee
Robert S. Pepper Distinguished Professor, UC Berkeley
EECS 290n – Advanced Topics in Systems Theory
Concurrent Models of Computation
Spring 2009

Week 9: Scheduling Dataflow Models

Execution Policy for a Dataflow Actor



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

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

Then the output of the actor is the concatenation of the results of a sequence of applications of the firing function:

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

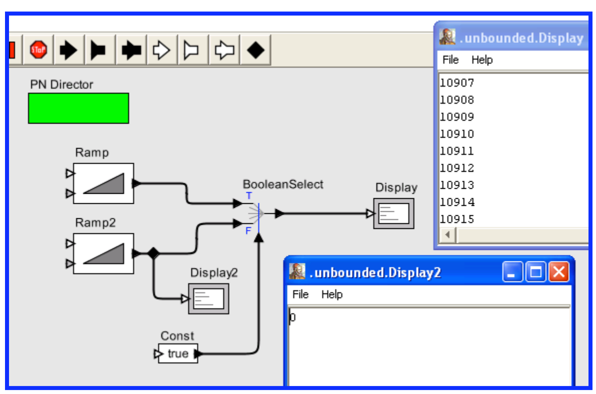
The problem we address now is *scheduling*: how to choose which actor to fire when there are choices.

Apply the Same Policy as for PN

- Define a *correct execution* to be any execution for which after any finite time every signal is a prefix of the LUB signal given by the semantics.
- Define a *useful execution* to be a correct execution that satisfies the following criteria:
  1. For every non-terminating PN model, after any finite time, a useful execution will extend at least one signal in finite (additional) time.
  2. If a correct execution satisfying criterion (1) exists that executes with bounded buffers, then a useful execution will execute with bounded buffers.

Policies that Fail

- Fair scheduling
- Demand driven
- Data driven



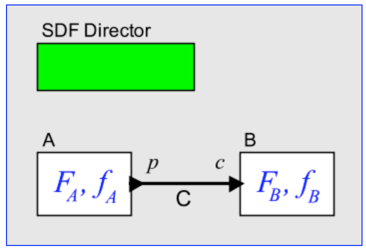
Adapting Parks' Strategy to Dataflow

- Require that the scheduler "know" how many tokens a firing will produce on each output port before that firing is invoked.
- Start with an arbitrary bound on the capacity of all buffers.
- Execute enabled actors that will not overflow the buffers on their outputs.
- If deadlock occurs and at least one actor is blocked on a enabled, increase the capacity of at least one buffer to allow an actor to fire.
- Continue executing, repeatedly checking for deadlock.

But Often the Firing Sequence can be Statically Determined! A History of Attempts:

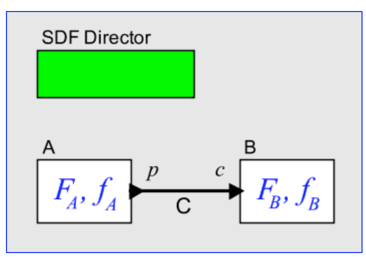
- Computation graphs [Karp & Miller - 1966]
- Process networks [Kahn - 1974]
- Static dataflow [Dennis - 1974]
- Dynamic dataflow [Arvind, 1981]
- K-bounded loops [Culler, 1986]
- Synchronous dataflow [Lee & Messerschmitt, 1986]
- Structured dataflow [Kodosky, 1986] now
- PGM: Processing Graph Method [Kaplan, 1987]
- Synchronous languages [Lustre, Signal, 1980's]
- Well-behaved dataflow [Gao, 1992]
- Boolean dataflow [Buck and Lee, 1993]
- Multidimensional SDF [Lee, 1993]
- Cyclo-static dataflow [Lauwereins, 1994]
- Integer dataflow [Buck, 1994]
- Bounded dynamic dataflow [Lee and Parks, 1995]
- Heterochronous dataflow [Girault, Lee, & Lee, 1997]
- Parameterized dataflow [Bhattacharya and Bhattacharyya 2001]
- Structured dataflow (again) [Thies et al. 2002]
- …

Synchronous Dataflow – SDF
(not to be confused with SR models!)



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.

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.

Relating to Infinite Firings

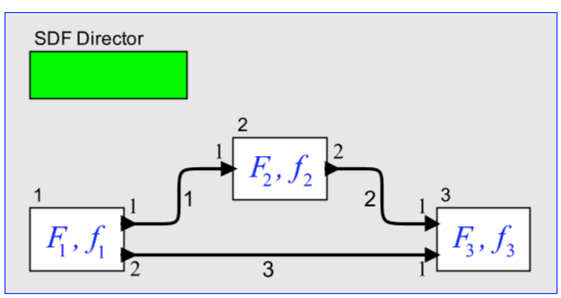
Of course, if $q_A = q_B = \infty$, then the balance equations are trivially satisfied.

By keeping a system in balance as an infinite execution proceeds, we can keep the buffers bounded.

Whether we can have a bounded infinite execution turns out to be decidable for SDF models.

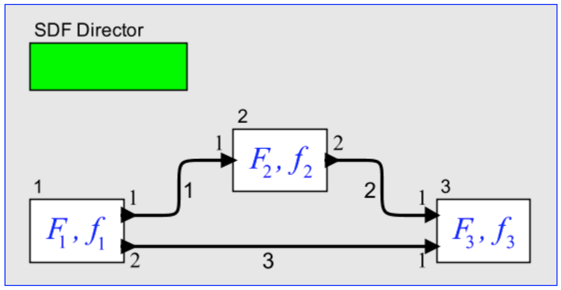
Example

Consider this example, where actors and arcs are numbered:



The balance equations imply that actor 3 must fire twice as often as the other two actors.

Compactly Representing the Balance Equations



production/consumption matrix

Connector 1

$$\Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & -1 \\ 2 & 0 & -1 \end{bmatrix}$$

Actor 1

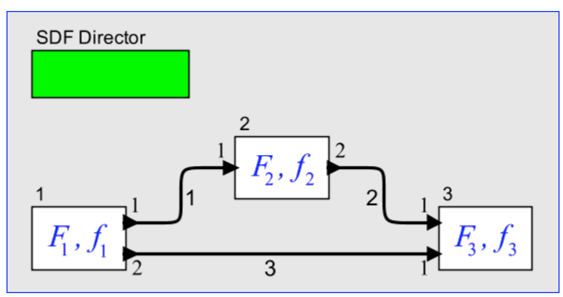
$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

firing vector

balance equations

$$\Gamma q = \vec{0} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Example

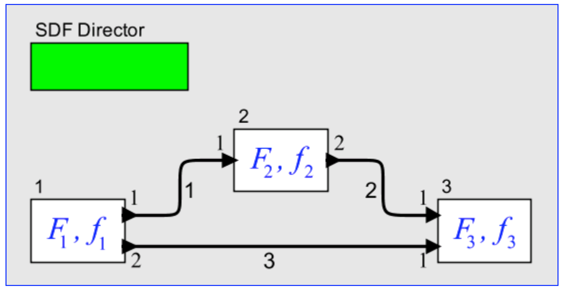


A solution to balance equations:

$$q = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \qquad \Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 2 & -1 \\ 2 & 0 & -1 \end{bmatrix} \qquad \Gamma q = \vec{0}$$

This tells us that actor 3 must fire twice as often as actors 1 and 2.

Example

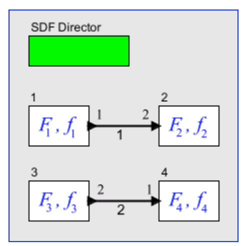


But there are many solutions to the balance equations:

$$q = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \quad q = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad q = \begin{bmatrix} 2 \\ 2 \\ 4 \end{bmatrix} \quad q = \begin{bmatrix} -1 \\ -1 \\ -2 \end{bmatrix} \quad q = \begin{bmatrix} \pi \\ \pi \\ 2\pi \end{bmatrix} \qquad \Gamma q = \vec{0}$$

We will see that for "well-behaved" models, there is a unique least positive solution.

Disconnected Models



For a disconnected model with two connected components, solutions to the balance equations have the form:

Solutions are linear combinations of the solutions for each connected component:

$$\Gamma = \begin{bmatrix} 1 & -2 & 0 & 0 \\ 0 & 0 & 2 & -1 \end{bmatrix} \qquad q = \begin{bmatrix} 2n \\ n \\ m \\ 2m \end{bmatrix} = n \begin{bmatrix} 2 \\ 1 \\ 0 \\ 0 \end{bmatrix} + m \begin{bmatrix} 0 \\ 0 \\ 1 \\ 2 \end{bmatrix}$$

$$\Gamma q = \vec{0}$$

Disconnected Models are Just Separate Connected Models

Define a *connected model* to be one where there is a path from any actor to any other actor, and where every connection along the path has production and consumption numbers greater than zero.

It is sufficient to consider only connected models, since disconnected models are disjoint unions of connected models. A schedule for a disconnected model is an arbitrary interleaving of schedules for the connected components.

Least Positive Solution to the Balance Equations

Note that if p_C, c_C , the number of tokens produced and consumed on a connection C, are non-negative integers, then the balance equation,

$$q_A p_C = q_B c_C$$

implies:

- q_A is rational if an only if q_B is rational.
- q_A is positive if an only if q_B is positive.

Consequence: Within any connected component, if there is any solution to the balance equations, then there is a unique least positive solution.

Rank of a Matrix

The rank of a matrix Γ is the number of linearly independent rows or columns. The equation

$$\Gamma q = \vec{0}$$

is forming a linear combination of the columns of G. Such a linear combination can only yield the zero vector if the columns are linearly dependent (this is what is means to be linearly dependent).

If Γ has a rows and b columns, the rank cannot exceed $\min(a, b)$. If the columns or rows of Γ are re-ordered, the resulting matrix has the same rank as Γ.

Rank of the Production/Consumption Matrix

Let a be the number of actors in a connected graph. Then the *rank* of the production/consumption matrix Γ must be a or $a - 1$.

Γ has a columns and at least $a - 1$ rows. If it has only $a - 1$ columns, then it cannot have rank a.

If the model is a *spanning tree* (meaning that there are barely enough connections to make it connected) then Γ has a rows and $a - 1$ columns. Its rank is $a - 1$. (Prove by induction).

Consistent Models

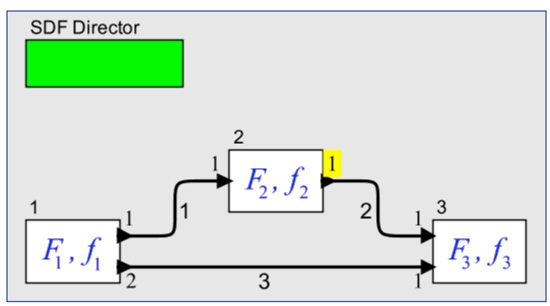
Let a be the number of actors in a connected model. The model is *consistent* if Γ has rank a - 1.

If the rank is a, then the balance equations have only a trivial solution (zero firings).

When Γ has rank a - 1, then the balance equations always have a non-trivial solution.

Example of an Inconsistent Model: No Non-Trivial Solution to the Balance Equations

$$\Gamma = \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ 2 & 0 & -1 \end{bmatrix}$$



This production/consumption matrix has rank 3, so there are no nontrivial solutions to the balance equations.

Dynamics of Execution

Consider a model with 3 actors. Let the *schedule* be a sequence $v : N_0 \to B^3$ where $B = \{0, 1\}$ is the binary set. That is,

$$v(n) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

to indicate firing of actor 1, 2, or 3.

Lee 09: 21

Buffer Sizes and Periodic Admissible Sequential Schedules (PASS)

Assume there are m connections and let $b : N_0 \to N^m$ indicate the buffer sizes prior to the each firing. That is, $b(0)$ gives the initial number of tokens in each buffer, $b(1)$ gives the number after the first firing, etc. Then

$$b(n+1) = b(n) + \Gamma v(n)$$

A *periodic admissible sequential schedule* (PASS) of length K is a sequence

$$v(0) \dots v(K-1)$$

such that $b(n) \geq \vec{0}$ for each $n \in \{0, \dots K-1\}$, and

$$b(K) = b(0) + \Gamma\left[v(0) + \dots + v(K-1)\right] = b(0)$$

Lee 09: 22

Periodic Admissible Sequential Schedules

Let $q = v(0) + \ldots + v(K-1)$
and note that we require that $\Gamma q = \vec{0}$.

A PASS will bring the model back to its initial state, and hence it can be repeated indefinitely with bounded memory requires.

A necessary condition for the existence of a PASS is that the balance equations have a non-zero solution. Hence, a PASS can only exist for a consistent model.

SDF Theorem 1

We have proved:

For a connected SDF model with a actors, a *necessary* condition for the existence of a PASS is that the model be consistent.

SDF Theorem 2

We have also proved:

For a consistent connected SDF model with production/ consumption matrix Γ, we can find an integer vector q where every element is greater than zero such that

$$\Gamma q = \vec{0}$$

Furthermore, there is a unique least such vector q.

SDF Sequential Scheduling Algorithms

Given a consistent connected SDF model with production/consumption matrix Γ, find the least positive integer vector q such that $\Gamma q = \vec{0}$.

Let $K = \mathbf{1}^{\mathrm{T}} q$, where $\mathbf{1}^{\mathrm{T}}$ is a row vector filled with ones. Then for each of $n \in \{0, \ldots K-1\}$, choose a firing vector

$$v(n) = \left\{ \begin{bmatrix} 1 \\ 0 \\ \cdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ \cdots \\ 0 \end{bmatrix}, \ldots, \begin{bmatrix} 0 \\ 0 \\ \cdots \\ 1 \end{bmatrix} \right\}$$

The number of rows in $v(n)$ is a.

SDF Sequential Scheduling Algorithms (Continued)

.. such that $b(n+1) = b(n) + \Gamma v(n) \geq \vec{0}$ (each element is non-negative), where $b(0)$ is the initial state of the buffers, and

$$\sum_{n=0}^{K-1} v(n) = q$$

The resulting *schedule* ($v(0)$, $v(1)$, …, $v(K - 1)$) forms one cycle of an infinite periodic schedule.

Such an algorithm is called an *SDF Sequential Scheduling Algorithm (SSSA).*

SDF Theorem 3

If an SDF model has a correct infinite sequential execution that executes in bounded memory, then any SSSA will find a schedule that provides such an execution.

Proof outline: Must show that if an SDF has a correct, infinite, bounded execution, then it has a PASS of length K. See Lee & Messerschmit [1987]. Then must show that the schedule yielded by an SSSA is correct, infinite, and bounded (trivial).

Note that every SSSA terminates.

Creating a Scheduler

Given a connected SDF model with actors $A_1, \ldots, A_a$:

Step 1: Solve for a rational q. To do this, first let $q_1 = 1$. Then for each actor A_i connected to A_1, let $q_i = q_1 m/n$, where m is the number of tokens A_1 produces or consumes on the connection to A_i, and n is the number of tokens A_i produces or consumes on the connection to A_1. Repeat this for each actor A_j connected to A_i for which we have not already assigned a value to q_j. When all actors have been assigned a value q_j, then we have a found a rational vector q such that $\Gamma q = \vec{0}$.

Lee 09: 29

Creating a Scheduler (continued)

Step 2: Solve for the least integer q. Use Euclid's algorithm to find the least common multiple of the denominators for the elements of the rational vector q. Then multiply through by that least common multiple to obtain the least positive integer vector q such that

$$\Gamma q = \vec{0}$$

Let $K = \mathbf{1}^T q$.

Lee 09: 30

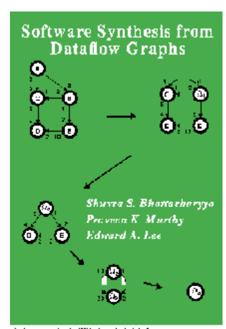
Creating a Scheduler (continued)

Step 3: For each $n \in \{0, \ldots, K-1\}$:

1. Given buffer sizes $b(n)$, determine which actors have firing rules that are satisfied (every source actor will have such a firing rule).
2. Select one of these actors that has not already been fired the number of times given by q. Let $v(n)$ be a vector with all zeros except in the position of the chosen actor, where its value is 1.
3. Update the buffer sizes:

$$b(n+1) = b(n) + \Gamma v(n)$$

A Key Question: If More Than One Actor is Fireable in Step 2, How do I Select One?

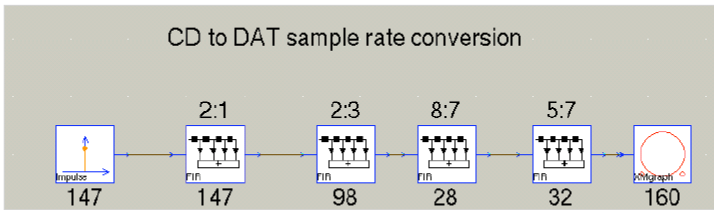


Optimization criteria that might be applied:

- Minimize buffer sizes.
- Minimize the number of actor activations.
- Minimize the size of the representation of the schedule (code size).

See S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs,* Kluwer Academic Press, 1996.

Minimum Buffer Schedule



A B A B C A B C A B A B C A B C D E A F F F F F B A B C A B C A B A B C D E
A F F F F F B C A B A B C A B C A B A B C D E A F F F F F B C A B A B C A B C
D E A F F F F F B A B C A B C A B A B C A B C D E A F F F F F B A B C A B C A
B A B C D E A F F F F F B C A B A B C A B C A B A B C D E A F F F F F E B C A
F F F F F B A B C A B C D E A F F F F F B A B C A B C A B A B C A B C D E A F
F F F F B A B C A B C A B A B C D E A F F F F F B C A B A B C A B C A B A B C
D E A F F F F F B C A B A B C A B C D E A F F F F F B A B C A B C A B A B C A
B C D E A F F F F F B A B C A B C A B A B C D E A F F F F F E B C A F F F F F B
A B C A B C A B A B C D E A F F F F F B C A B A B C A B C D E A F F F F F B A
B C A B C A B A B C A B C D E A F F F F F B A B C A B C A B A B C D E A F F F
F F B C A B A B C A B C A B A B C D E A F F F F F B C A B A B C A B C D E A F
F F F F B A B C A B C A B A B C A B C D E A F F F F F F E B A F F F F F B C A B C
A B A B C D E A F F F F F B C A B A B C A B C A B A B C D E A F F F F F B C A
B A B C A B C D E A F F F F F B A B C A B C A B A B C A B C D E A F F F F F B
A B C A B C A B A B C D E A F F F F F B C A B A B C A B C A B A B C D E A F
F F F F B C A B A B C A B C D E F F F F F F E F F F F F

Source: Shuvra Bhattacharyya

Code Generation (Circa 1992)

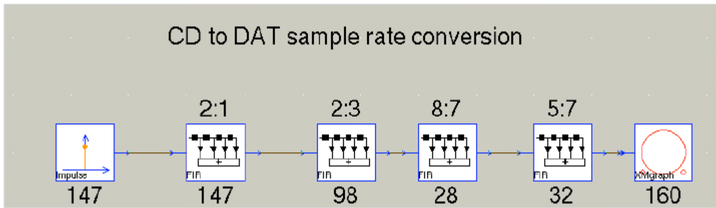
Block specification for DSP code generation in Ptolemy Classic:

```
        codeblock(std) {
        ; initialize address registers for coef and
 delayLineove     #$addr(coef)+$val(coefLen)-1,r3
; insert here
        move      $ref(delayLineStart),r5
; delayLine
        move      #$val(stepSize),x1
        move      $ref(error),x0
        mpyr      x0,x1,a
        move      a,x0
        move      x:(r3),b           y:(r5)+,y0
        }

        codeblock(loop) {
        do        #$val(loopVal),$label(endloop)
        macr      x0,y0,b
        move      b,x:(r3)-
        move      x:(r3),b           y:(r5)+,y0
$label(endloop)
        }

        codeblock(noloop) {
        macr      x0,y0,b
        move      b,x:(r3)-
        move      x:(r3),b           y:(r5)+,y0
        }
```

macros defined by the code generator

alternative code blocks chosen based on parameter values

Scheduling Tradeoffs
(Bhattacharyya, Parks, Pino)



Scheduling strategy	Code	Data
Minimum buffer schedule, no looping	13735	32
Minimum buffer schedule, with looping	9400	32
Worst minimum code size schedule	170	1021
Best minimum code size schedule	170	264

Source: Shuvra Bhattacharyya

Parallel Scheduling

It is easy to create an SSSA that as it produces a PASS, it constructs an *acyclic precedence graph* (APG) that represents the dependencies that an actor firing has on prior actor firings.

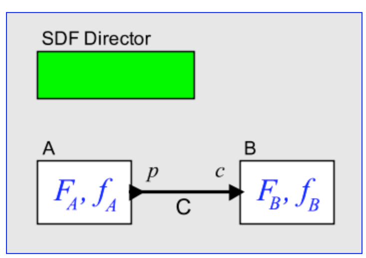
Given such an APG, the parallel scheduling problem is a standard one where there are many variants of the optimization criteria and scheduling heuristics.

See many papers on the subject on the Ptolemy website.

Taking Stock

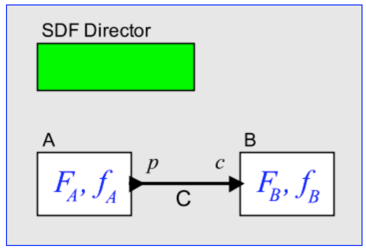
- SDF models have actors that produce and consume a fixed (constant) number of tokens on each arc.
- A periodic admissible sequential schedule (PASS) is a finite sequence of firings that brings buffers back to their initial state and keeps buffer sizes non-negative.
- A necessary condition for the existence of a PASS is that the balance equations have a non-trivial solution.
- A class of algorithms has been identified that will always find a PASS if one exists.

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.

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.

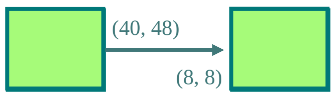
Extensions of SDF that Improve Expressiveness

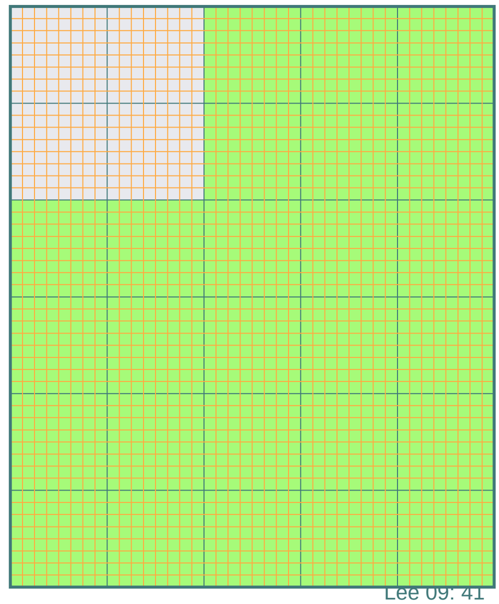
Structured Dataflow [Kodosky 86, Thies et al. 02]
Boolean dataflow [Buck and Lee, 93]
Cyclostatic Dataflow [Lauwereins 94]
Multidimensional SDF [Lee & Murthy 96]
Heterochronous Dataflow [Girault, Lee, and Lee, 97]
Parameterized Dataflow [Bhattacharya et al. 00]
Teleport Messages [Thies et al. 05]

Many of these remain decidable

Multidimensional SDF
(Lee, 1993)

Production and consumption of *N*-dimensional arrays of data:





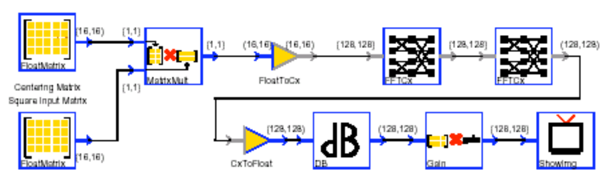
Balance equations and scheduling policies generalize.

Much more data parallelism is exposed.

Similar (but dynamic) multidimensional streams have been implemented in Lucid.

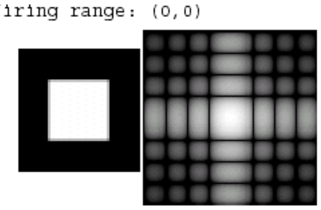
More interesting Example

Two dimensional FFT constructed out of one-dimensional actors.

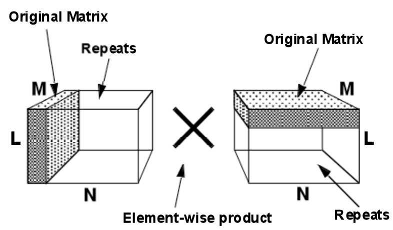


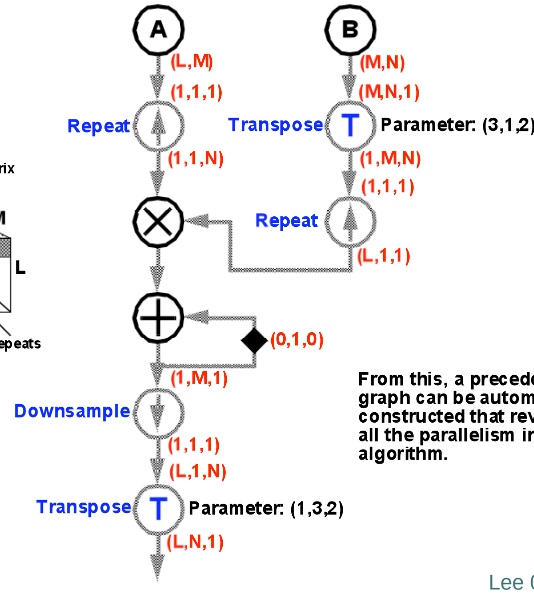

```
MDSDF SCHEDULE:
fft_of_square2.FloatMatrix1, firing range: (0,0)
fft_of_square2.FloatMatrix2, firing range: (0,0)
fft_of_square2.Mult1, firing range: (0,0) - (15,15)
fft_of_square2.FloatToCx1, firing range: (0,0)
fft_of_square2.FFTCx2, firing range: (0,0) - (15,0)
fft_of_square2.FFTCx1, firing range: (0,0) - (0,127)
fft_of_square2.CxToFloat1, firing range: (0,0)
fft_of_square2.DB1, firing range: (0,0)
fft_of_square2.Gain1, firing range: (0,0)
fft_of_square2.ShowImg1, firing range: (0,0)
```

Figure 6. Screen dump of 2D-FFT system, the associated schedule, and outputs.



MDSDF Structure Exposes Fine-Grain Data Parallelism





From this, a precedence graph can be automatically constructed that reveals all the parallelism in the algorithm.

However, such programs are extremely hard to write (and to read).

Extensions of MDSDF

Extended to non-rectangular lattices and connections to number theory:

P. K. Murthy, "Scheduling Techniques for Synchronous and Multidimensional Synchronous Dataflow," Technical Memorandum UCB/ERL M96/79, Ph.D. Thesis, EECS Department, University of California, Berkeley, CA 94720, December 1996.

Praveen K. Murthy and Edward A. Lee, "Multidimensional Synchronous Dataflow ," *IEEE Transactions on Signal Processing*, volume 50, no. 8, pp. 2064 -2079, July 2002.

Extensions of SDF that Improve Expressiveness

Structured Dataflow [Kodosky 86, Thies et al. 02]
Boolean dataflow [Buck and Lee, 93]
Cyclostatic Dataflow [Lauwereins 94]
Multidimensional SDF [Lee & Murthy 96]
Heterochronous Dataflow [Girault, Lee, and Lee, 97]
Parameterized Dataflow [Bhattacharya et al. 00]
Teleport Messages [Thies et al. 05]

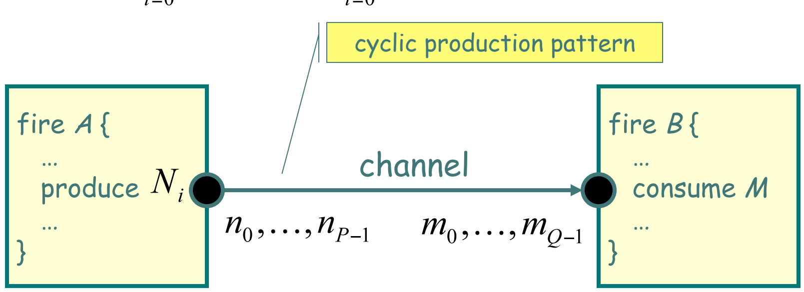
Many of these remain decidable

Cyclostatic Dataflow (CSDF)

(Lauwereins et al., TU Leuven, 1994)

Actors cycle through a regular production/consumption pattern.
Balance equations become:

$$q_A \sum_{i=0}^{R-1} n_{i \bmod P} = q_B \sum_{i=0}^{R-1} m_{i \bmod Q}; \; R = lcm(P, Q)$$

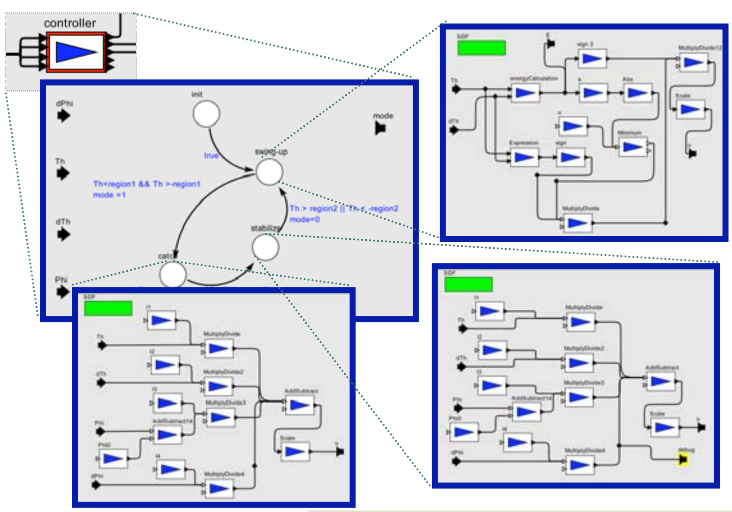


Extensions of SDF that Improve Expressiveness

Structured Dataflow [Kodosky 86, Thies et al. 02]
Boolean dataflow [Buck and Lee, 93]
Cyclostatic Dataflow [Lauwereins 94]
Multidimensional SDF [Lee & Murthy 96]
Heterochronous Dataflow [Girault, Lee, and Lee, 97]
Parameterized Dataflow [Bhattacharya et al. 00]
Teleport Messages [Thies et al. 05]

Many of these remain decidable

Heterochronous Dataflow (HDF)
(Girault, Lee, & Lee, 1997)



An actor consists of a state machine and refinements to the states that define behavior.

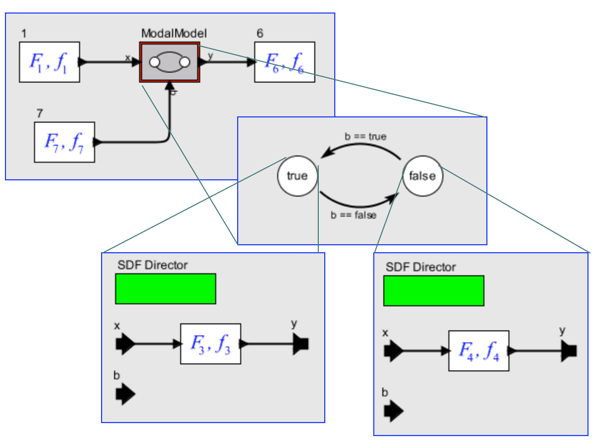
Heterochronous Dataflow (HDF)
(Girault, Lee, and Lee, 1997)

Related to "parameterized dataflow" of Bhattacharya and Bhattacharyya (2001).

- An interconnection of actors.
- An actor is either SDF or HDF.
- If HDF, then the actor has:
  - a state machine
  - a refinement for each state
  - where the refinement is an SDF or HDF actor
- Operational semantics:
  - with the state of each state machine fixed, graph is SDF
  - in the initial state, execute one complete SDF iteration
  - evaluate guards and allow state transitions
  - in the new state, execute one complete SDF iteration
- HDF is decidable if state machines are finite
  - but complexity can be high

Lee 09: 49

If-Then-Else Using Heterochronous Dataflow



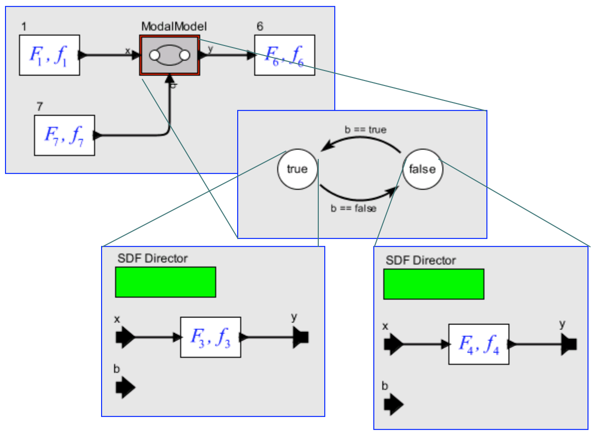
Imperative equivalent:

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

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

Lee 09: 50

If-Then-Else Using Heterochronous Dataflow

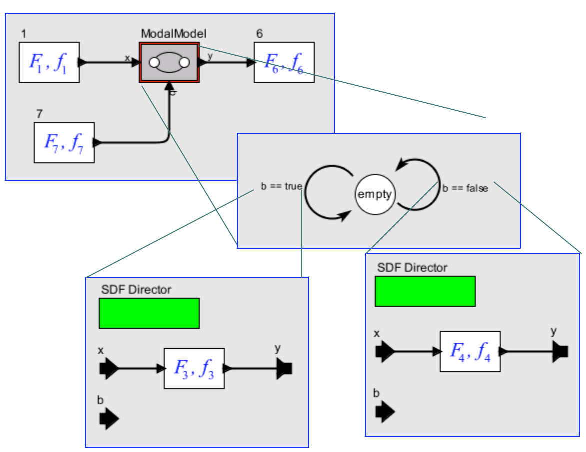


Imperative equivalent:

```
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.

Hierarchical SDF Using Transition Refinements



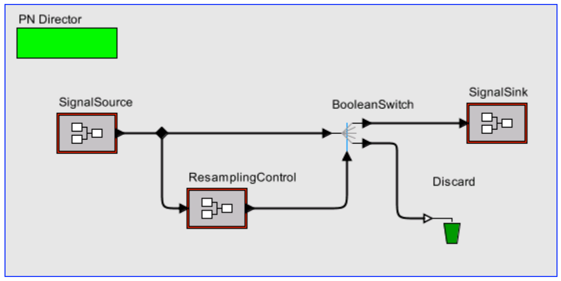
Imperative equivalent:

```
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.

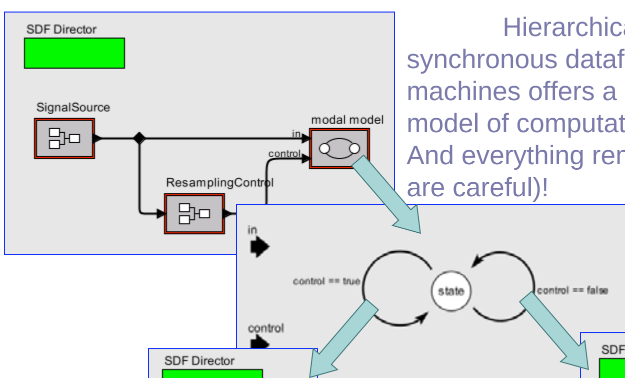
Application of Dynamic Dataflow: Resampling of Streaming Media



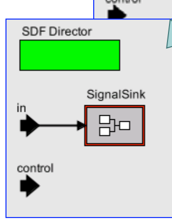
This pattern requires the use of a semantically richer dataflow model than SDF because the BooleanSwitch is not an SDF actor.

This has a performance cost and reduces the static analyzability of the model.

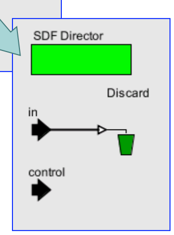
Resampling Design Pattern using Modal Models



Hierarchically mixing synchronous dataflow with finite state machines offers a much more powerful model of computation than either alone. And everything remains decidable (if you are careful)!



This generalizes structured dataflow with sequential decision logic, but without the cost of undecidability.



Taking Stock

- Generalizations to SDF improve expressiveness while preserving decidability.
- Usable languages for many of these extensions have yet to be created.

Lee 09: 55

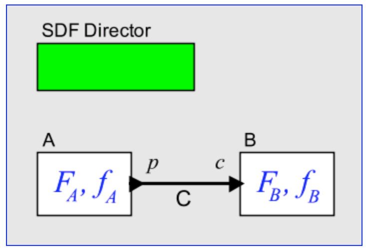
Extensions of SDF that Improve Expressiveness

Structured Dataflow [Kodosky 86, Thies et al. 02]
(the other) Synchronous Dataflow [Halbwachs et al. 91]
Boolean dataflow [Buck and Lee, 93]
Cyclostatic Dataflow [Lauwereins 94]
Multidimensional SDF [Lee & Murthy 96]
Heterochronous Dataflow [Girault, Lee, and Lee, 97]
Parameterized Dataflow [Bhattacharya et al. 00]
Teleport Messages [Thies et al. 05]

Many of these remain decidable

Lee 09: 56

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.

Expressiveness Limitations in SDF

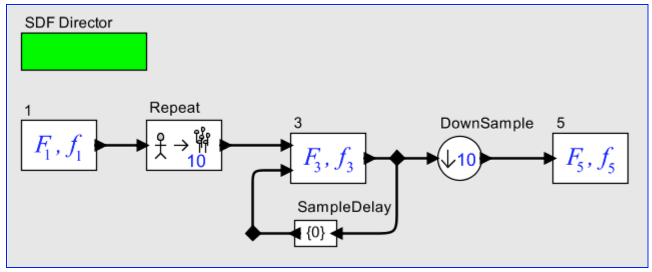
SDF cannot express data-dependent flow of tokens:

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

Hierarchical SDF 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 SDF.

Manifest Iteration in SDF

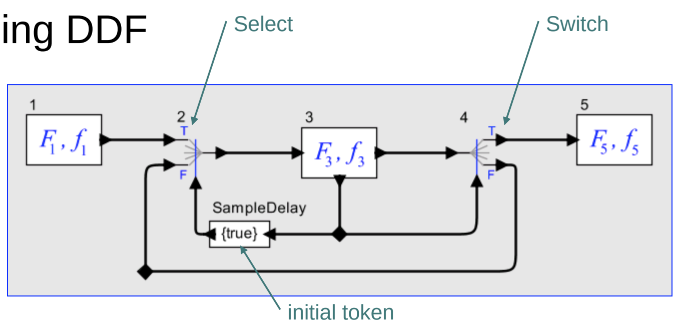


Imperative equivalent:

```
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 SDF. But data-dependent iteration is not.

Do-While Using DDF



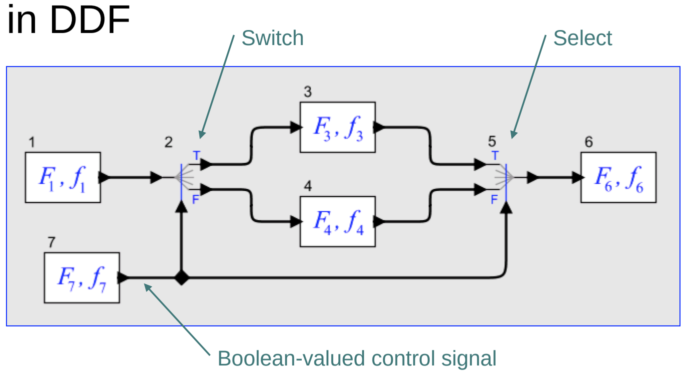
Imperative equivalent:

```
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 SDF?

If-Then-Else in DDF

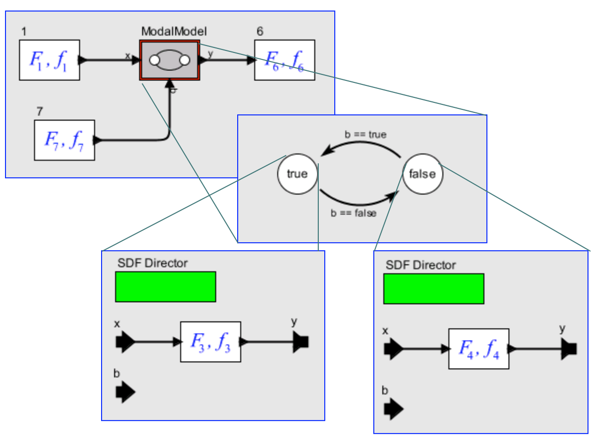


Imperative equivalent:

```
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.

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

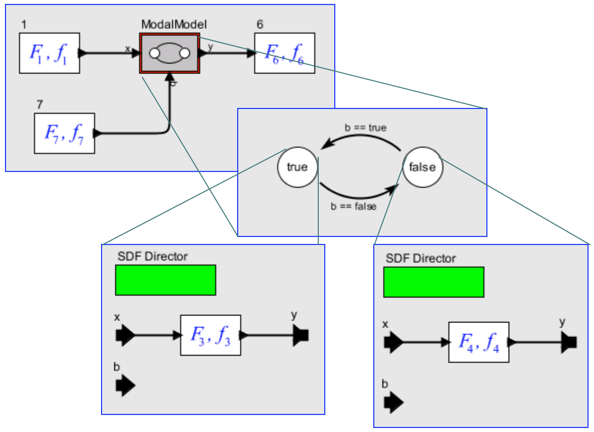


Imperative equivalent:

```
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.

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

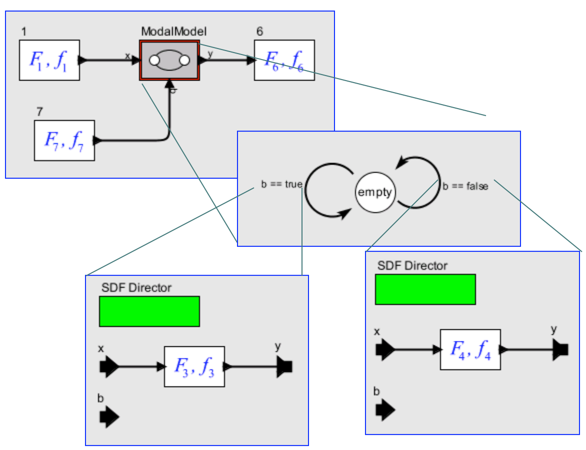


Imperative equivalent:

```
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.

Hierarchical SDF Using Transition Refinements



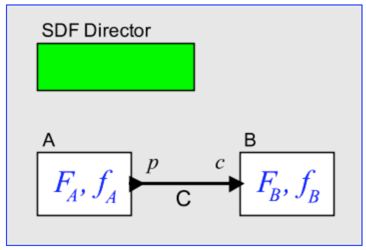
Imperative equivalent:

```
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.

Balance Equations



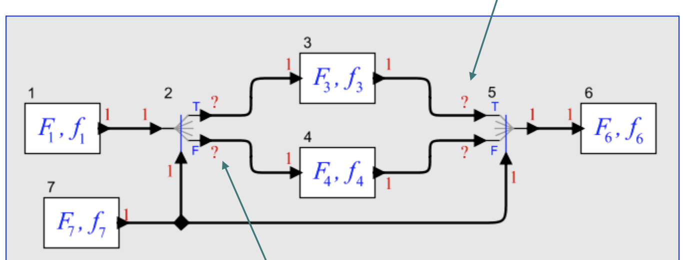
Let q_A, q_B be the number of firings of actors A and B.
Let p_C, c_C be the number of token produced and consumed on a connection C.
Then the system is *in balance* if for all connections C

$$q_A p_C = q_B c_C$$

where A produces tokens on C and B consumes them.

If-Then-Else in DDF

What consumption rate?



What production rate?

Imperative equivalent:

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

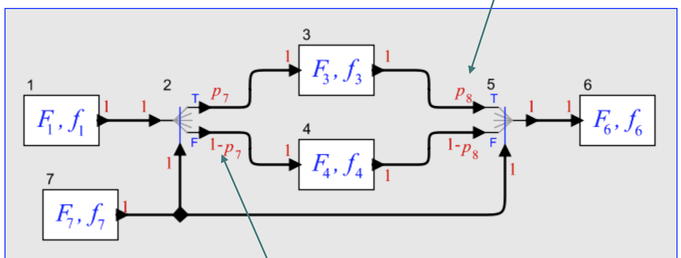
The if-then-else model is not SDF. But we can clearly give a bounded *quasi-static* schedule for it:

(1, 7, 2, b?3, !b?4, 5, 6)

guard

Symbolic Rates

Symbolic consumption rate.



Imperative equivalent:

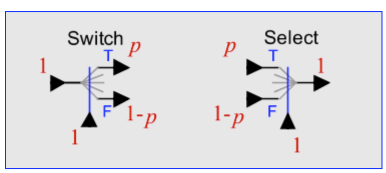
```
while (true) {
   x = f1();
   b = f7();
   if (b) {
     y = f3(x);
   } else {
     y = f4(x);
   }
   f6(y);
}
```

Symbolic production rate.

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

Lee 09: 67

Interpretations of Symbolic Rates

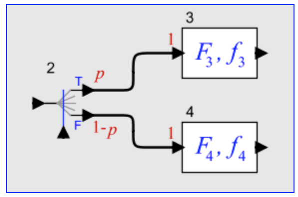


- General interpretation: p is a symbolic placeholder for an unknown.
- Probabilistic interpretation: p is the probability that a Boolean control input is *true.*
- Proportion interpretation: p is the proportion of *true* values at the control input in one *complete cycle*.

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

Lee 09: 68

Symbolic Balance Equations



The two connections above imply the following balance equations:

$$q_2 \, p = q_3$$

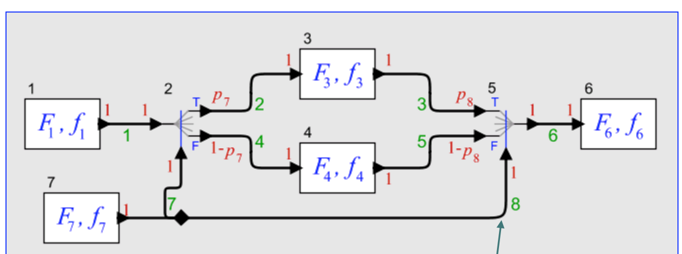
$$q_2 \, (1 - p) = q_4$$

Lee 09: 69

Symbolic Rates

Imperative equivalent:

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

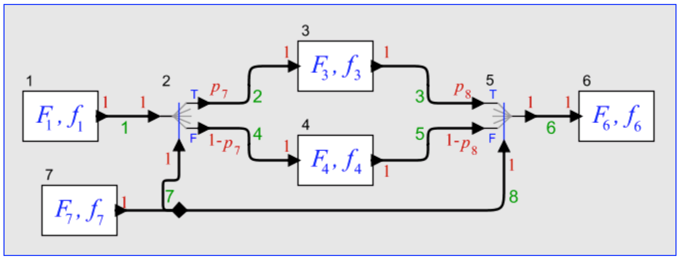


Label the arcs

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

Lee 09: 70

Production/Consumption Matrix for If-Then-Else



Symbolic variables:

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

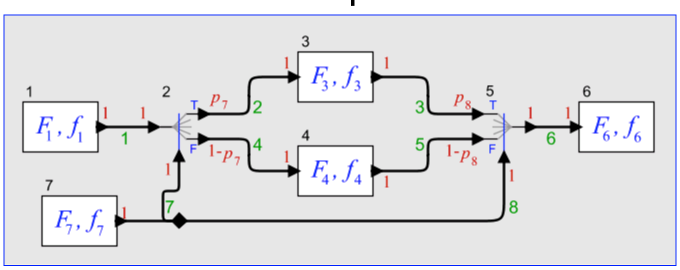
$$\Gamma(\vec{p}) = \begin{bmatrix} 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & p_7 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & -p_8 & 0 & 0 \\ 0 & 1-p_7 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -(1-p_8) & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 & 1 \end{bmatrix}$$

Balance equations:

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

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

Production/Consumption Matrix for If-Then-Else



The balance equations have a solution $q(\vec{p})$ if an 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.

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

Strong and Weak Consistency

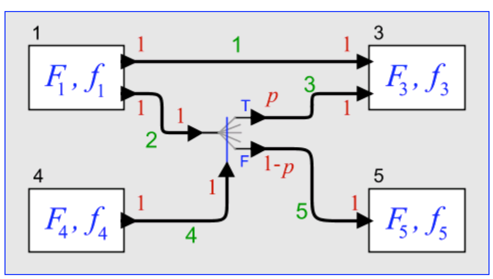
A *strongly consistent* dataflow model is one where the balance equations have a solution that is provably valid without concern for the values of the symbolic variables.

- 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.

Weakly Consistent Model

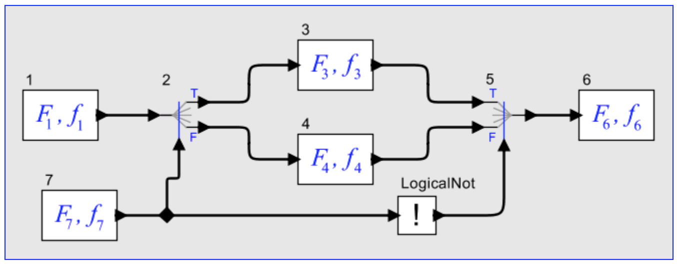


$$\Gamma(\vec{p}) = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 \\ 0 & p & -1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 \\ 0 & 1-p & 0 & 0 & -1 \end{bmatrix}$$

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

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

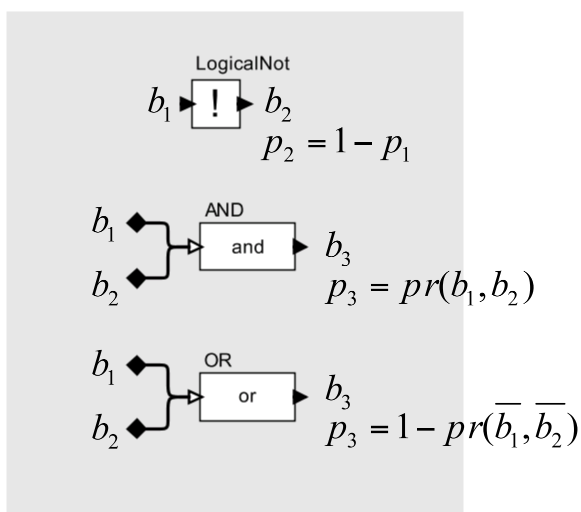
Another Example of a Weakly Consistent Model



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.

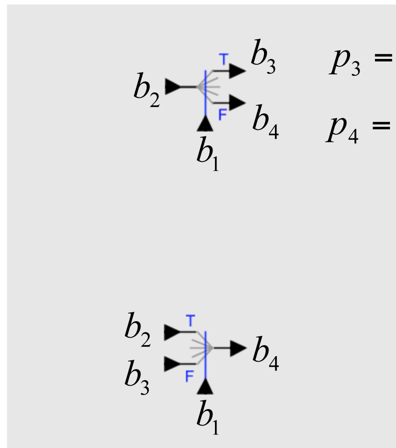
Use Boolean Relations

Symbolic variables across logical operators can be related as shown.



Routing of Boolean Tokens

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



$$p_3 = pr(b_2 \mid b_1)$$

$$p_4 = pr(b_2 \mid \overline{b_1})$$

$$p_4 = pr(b_2 \mid b_1) + pr(b_3 \mid \overline{b_1})$$

Lee 09: 77

Taking Stock

- 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 09: 78