Dataflow Networks

- Dataflow Networks
- Syntax and Semantics
  - actor, tokens and firing
- Static scheduling
- Other dataflow models

Data-flow networks

- Powerful formalism for data-dominated system specification
- Partially-ordered model (no over-specification)
- Deterministic execution independent of scheduling
- Used for
  - simulation
  - scheduling
  - memory allocation
  - code generation
for Digital Signal Processors (HW and SW)
Dataflow network

- A data-flow network is a collection of functional nodes which are connected and communicate over unbounded FIFO queues
- Nodes are commonly called actors
- The bits of information that are communicated over the queues are commonly called tokens

Intuitive Semantics

- Unbounded FIFOs perform communication via sequences of tokens carrying values
  - integer, float, fixed point
  - matrix of integer, float, fixed point
  - image of pixels
- State implemented as self-loop
- Determinacy:
  - unique output sequences given unique input sequences
  - Sufficient condition: blocking read
    (process cannot test input queues for emptiness)
Intuitive semantics

- At each time, one actor is fired
- When firing, actors consume input tokens and produce output tokens
- Actors can be fired only if there are enough tokens in the input queues

Example: FIR filter

- single input sequence \( i(n) \)
- single output sequence \( o(n) \)
- \( o(n) = c1 \cdot i(n) + c2 \cdot i(n-1) \)
Intuitive semantics

- Example: FIR filter
  - single input sequence \( i(n) \)
  - single output sequence \( o(n) \)
  - \( o(n) = c1 \cdot i(n) + c2 \cdot i(n-1) \)
Intuitive semantics

- Example: FIR filter
  - single input sequence \( i(n) \)
  - single output sequence \( o(n) \)
  - \( o(n) = c_1 i(n) + c_2 i(n-1) \)

\[
\begin{align*}
  i & \rightarrow c_1 \rightarrow c_2 \\
  i & \rightarrow c_1 \rightarrow + \rightarrow o
\end{align*}
\]
Intuitive semantics

- Example: FIR filter
  - single input sequence \( i(n) \)
  - single output sequence \( o(n) \)
  - \( o(n) = c_1 i(n) + c_2 i(n-1) \)
Intuitive semantics

- Example: FIR filter
  - single input sequence $i(n)$
  - single output sequence $o(n)$
  - $o(n) = c1 \cdot i(n) + c2 \cdot i(n-1)$

slide: ASV-UCB
Intuitive semantics

- Example: FIR filter
  - single input sequence \( i(n) \)
  - single output sequence \( o(n) \)
  - \( o(n) = c_1 i(n) + c_2 i(n-1) \)

Questions

- Does the order in which actors are fired affect the final result?
- Does it affect the “operation” of the network in any way?
- Go to Radio Shack and ask for an unbounded queue!!
Formal semantics: sequences

- Actors operate from a sequence of input tokens to a sequence of output tokens.
- Let tokens be noted by $x_1$, $x_2$, $x_3$, etc...
- A sequence of tokens is defined as $X = [x_1, x_2, x_3, ...]$
- Over the execution of the network, each queue will grow a particular sequence of tokens.
- In general, we consider the actors mathematically as functions from sequences to sequences (not from tokens to tokens).

Ordering of sequences

- Let $X_1$ and $X_2$ be two sequences of tokens.
- We say that $X_1$ is less than $X_2$ if and only if (by definition) $X_1$ is an initial segment of $X_2$.
- Homework: prove that the relation so defined is a partial order (reflexive, antisymmetric, and transitive).
- This is also called the prefix order.
- Example: $[x_1, x_2] \leq [x_1, x_2, x_3]$.
- Example: $[x_1, x_2]$ and $[x_1, x_3, x_4]$ are incomparable.
**Chains of sequences**

- Consider the set $S$ of all finite and infinite sequences of tokens
- This set is partially ordered by the prefix order
- A subset $C$ of $S$ is called a chain iff all pairs of elements of $C$ are comparable
- If $C$ is a chain, then it must be a linear order inside $S$ (hence the name chain)
- Example: $\{ [x_1], [x_1, x_2], [x_1, x_2, x_3], \ldots \}$ is a chain
- Example: $\{ [x_1], [x_1, x_2], [x_1, x_3], \ldots \}$ is not a chain

**(Least) Upper Bound**

- Given a subset $Y$ of $S$, an upper bound of $Y$ is an element $z$ of $S$ such that $z$ is larger than all elements of $Y$
- Consider now the set $Z$ (subset of $S$) of all the upper bounds of $Y$
- If $Z$ has a least element $u$, then $u$ is called the least upper bound (lub) of $Y$
- The least upper bound, if it exists, is unique
- Note: $u$ might not be in $Y$ (if it is, then it is the largest value of $Y$)
Complete Partial Order

■ Every chain in S has a least upper bound
■ Because of this property, S is called a Complete Partial Order
■ Notation: if C is a chain, we indicate the least upper bound of C by lub( C )
■ Note: the least upper bound may be thought of as the limit of the chain

Processes

■ Process: function from a p-tuple of sequences to a q-tuple of sequences
  \( F : S^p \rightarrow S^q \)
■ Tuples have the induced pointwise order:
  \( Y = (y_1, \ldots, y_p), Y' = (y'_1, \ldots, y'_p) \) in \( S^p \):
  \( Y \preceq Y' \) iff \( y_i \preceq y'_i \) for all \( 1 \leq i \leq p \)
■ Given a chain \( C \) in \( S^p \), \( F( C ) \) may or may not be a chain in \( S^q \)
■ We are interested in conditions that make that true
Continuity and Monotonicity

- Continuity: F is continuous iff (by definition) for all chains C, \( \text{lub}(F(C)) \) exists and
  \[ F(\text{lub}(C)) = \text{lub}(F(C)) \]
- Similar to continuity in analysis using limits
- Monotonicity: F is monotonic iff (by definition) for all pairs X, X'
  \[ X \leq X' \Rightarrow F(X) \leq F(X') \]
- Continuity implies monotonicity
  - intuitively, outputs cannot be “withdrawn” once they have been produced
  - timeless causality. F transforms chains into chains

Summary of function class

- Summary of function class and their relationship for the function F: \( S^p \rightarrow S^q \)

Sequential \( \Rightarrow \) continuous \( \Rightarrow \) monotonic
Non-monotonic processes

- "Canonical" non-monotonic process: *fair merge*

\[
[x_1, x_2, x_3, \ldots] \rightarrow [x_1, y_1, x_2, y_2, x_3, y_3, \ldots]
\]
\[
[y_1, y_2, y_3, \ldots] \rightarrow [x_1, y_1, x_2, y_2, y_3, \ldots]
\]
\[
[x_1, x_2] \rightarrow [x_1, y_1, x_2, y_2, x_3, \ldots]
\]
\[
[y_1, y_2, y_3, \ldots] \rightarrow [x_1, y_1, x_2, y_2, y_3, \ldots]
\]
\[
[x_1, x_2, x_3, \ldots] \rightarrow [x_1, y_1, x_2, y_2, x_3, \ldots]
\]
\[
[y_1, y_2] \rightarrow [x_1, y_1, x_2, y_2, x_3, \ldots]
\]

In the previous example, we have:

\[(x_1, x_2), [y_1, y_2, y_3, \ldots] \leq ([x_1, x_2, x_3, \ldots], [y_1, y_2, y_3, \ldots])\]

- but \([x_1, y_1, x_2, y_2, x_3, y_3, \ldots]\) and \([x_1, y_1, x_2, y_2, y_3, \ldots]\) are incomparable.

- The process is not monotonic (needs prediction of the future to be really fair).
From Kahn networks to Data-flow networks

- Each process becomes an *actor*: set of pairs of
  - firing rule
    (number of required tokens on inputs)
  - function
    (including number of consumed and produced tokens)
- Formally shown to be equivalent, but actors with firing are more intuitive
- *Mutually exclusive* firing rules imply monotonocity
- Generally simplified to *blocking read*

Examples of Data-flow actors

- **SDF**: Synchronous (or, better, Static) Data-flow
  - fixed input and output tokens

  ![Diagram for SDF]

- **BDF**: Boolean Data-flow
  - control token determines consumed and produced tokens

  ![Diagram for BDF]
Static scheduling of DF

- Key property of DF networks: output sequences do not depend on time of firing of actors
- SDF networks can be statically scheduled at compile-time
  - execute an actor when it is known to be fireable
  - no overhead due to sequencing of concurrency
  - static buffer sizing
- Different schedules yield different
  - code size
  - buffer size

Static scheduling of SDF

- Based only on process graph (ignores functionality)
- Network state: number of tokens in FIFOs
- Objective: find schedule that is valid, i.e.:
  - admissible
    - (only fires actors when fireable)
  - periodic
    - (brings network back to initial state by firing each actor at least once)
- Optimize cost function over admissible schedules
Balance equations

- Number of produced tokens must equal number of consumed tokens on every edge

- Repetitions (or firing) vector $v_S$ of schedule $S$: number of firings of each actor in $S$

- $v_S(A) \cdot n_p = v_S(B) \cdot n_c$
  must be satisfied for each edge

Balance for each edge:
- $3 \cdot v_S(A) - v_S(B) = 0$
- $v_S(B) - v_S(C) = 0$
- $2 \cdot v_S(A) - v_S(C) = 0$
- $2 \cdot v_S(A) - v_S(C) = 0$
Tagged Token DFM

- Arvind and Gostelow 80’s
- Each token has a tag, firing is enabled when tokens have matching tags.
  - No need of FIFO discipline in the channel
Boolean Dataflow

- Actors have Boolean control ports which may control input and output ports. All other ports obey SDF semantics.
- Ex. Switch (de-multiplexing), Select (Mux)
- Boolean control ports are read first and then other input ports.
- Switch and select can be used to data-dependent iteration.
- Allows conditional flow of data.

Example of general DF

- Merge streams of multiples of 2 and 3 in order (removing duplicates)

```c
a = get(A)
b = get(B)
forever {
    if (a > b) {
        put(O, a)
a = get(A)
    } else if (a < b) {
        put(O, b)
b = get(B)
    } else {
        put(O, a)
a = get(A)
b = get(B)
    }
}
```

- Deterministic merge (no “peeking”)
Summary of DF networks

- Advantages:
  - Easy to use (graphical languages)
  - Powerful algorithms for
    - verification (fast behavioral simulation)
    - synthesis (scheduling and allocation)
  - Explicit concurrency

- Disadvantages:
  - Efficient synthesis only for restricted models
  - (no input or output choice)
  - Cannot describe reactive control (blocking read)