Models Of Computation for reactive systems

- Main MOCs:
  - Communicating Finite State Machines
  - Data-flow Process Networks
  - Petri Nets
  - Discrete Event
  - Codesign Finite State Machines
- Main languages:
  - StateCharts
  - Esterel
  - Data-flow networks

Dataflow networks

- A bit of history
- Syntax and semantics
  - actors, tokens and firings
- Scheduling of Static Data-flow
  - static scheduling
  - code generation
  - buffer sizing
- Other Data-flow models
  - Boolean Data-flow
  - Dynamic Data-flow
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)

A bit of history

- Karp computation graphs (‘66): seminal work
- Kahn process networks (‘58): formal model
- Dennis Data-flow networks (‘75): programming language for MIT DF machine
- Several recent implementations
  - graphical:
    - Ptolemy (UCB), Khoros (U. New Mexico), Grape (U. Leuven)
    - SPW (Cadence), COSSAP (Synopsys)
  - textual:
    - Silage (UCB, Mentor)
    - Lucid, Haskell
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

- (Often stateless) actors perform computation
- 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 \(o(n)\)
- \(o(n) = c1 \times i(n) + c2 \times i(n-1)\)
Intuitive semantics

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

![Diagram of FIR filter](image)

**Intuitive semantics**

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

![Diagram of FIR filter](image)
Intuitive semantics

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

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

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 $x_1, x_2, x_3, \text{etc…}$
- A sequence of tokens is defined as
  \[ X = [x_1, x_2, x_3, \ldots] \]
- 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 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:
  \[ Y = ( y_1, \ldots, y_p ), \quad Y' = ( y'_1, \ldots, y'_p ) \text{ in } S^p : \]
  \[ Y \leq Y' \text{ iff } y_i \leq y'_i \text{ for all } 1 \leq i \leq p \]
- 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, 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