Least Fixed Point semantics

- Let $X$ be the set of all sequences
- A network is a mapping $F$ from the sequences to the sequences (where $I$ represents the input sequence):
  $$X = F( X, I )$$
- The behavior of the network is defined as the unique least fixed point of the equation
- If $F$ is continuous then the least fixed point exists
  $$\text{LFP} = \text{LUB}( \{ F^n( \perp, I ) : n \geq 0 \} )$$
Non-monotonic processes

- “Canonical” non-monotonic process: *fair merge*

\[ \text{FM} \]

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

\[ [x_1, y_1, x_2, y_2, y_3, \ldots] \]
\[ [x_1, y_1, x_2, y_2, x_3, y_3, \ldots] \]
\[ [x_1, y_1, x_2, y_2, y_3, \ldots] \]
\[ [x_1, y_1, x_2, y_2, x_3, \ldots] \]
Non-monotonic processes

- 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)
- The least fixed point may not exist
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 monotonicity
- Generally simplified to *blocking read*
Examples of Data-flow actors

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

- **BDF: Boolean Data-flow**
  - control token determines consumed and produced tokens
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
  - pipeline utilization
Static scheduling of SDF

- Based only on \textit{process graph} (ignores functionality)
- Network state: number of tokens in FIFOs
- Objective: find schedule that is \textit{valid}, i.e.:
  - \textit{admissible}
    (only fires actors when fireable)
  - \textit{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 $S$

  - $v_S(A) \ n_p = v_S(B) \ n_c$

  must be satisfied for each edge
Balance equations

- Balance for each edge:
  - $3 \, v_S(A) - v_S(B) = 0$
  - $v_S(B) - v_S(C) = 0$
  - $2 \, v_S(A) - v_S(C) = 0$
  - $2 \, v_S(A) - v_S(C) = 0$
Balance equations

\[ M \cdot \mathbf{S} = 0 \]

iff \( S \) is periodic

- Full rank (as in this case)
  - no non-zero solution
  - no periodic schedule

(too many tokens accumulate on A->B or B->C)

\[
M = \begin{bmatrix}
3 & -1 & 0 \\
0 & 1 & -1 \\
2 & 0 & -1 \\
2 & 0 & -1 \\
\end{bmatrix}
\]
Balance equations

- Non-full rank
  - infinite solutions exist (linear space of dimension 1)
- Any multiple of $q = \begin{bmatrix} 1 & 2 & 2 \end{bmatrix}^T$ satisfies the balance equations
- ABCBC and ABBCC are minimal valid schedules
- ABABBCBCC is non-minimal valid schedule
Static SDF scheduling

- Main SDF scheduling theorem (Lee ‘86):
  - A connected SDF graph with $n$ actors has a periodic schedule if its topology matrix $M$ has rank $n-1$
  - If $M$ has rank $n-1$ then there exists a unique smallest integer solution $q$ to
    \[ Mq = 0 \]

- Rank must be at least $n-1$ because we need at least $n-1$ edges (connectedness), providing each a linearly independent row

- Admissibility is not guaranteed, and depends on initial tokens on cycles
Admissibility of schedules

- No admissible schedule: BACBA, then deadlock…
- Adding one token (delay) on A->C makes BACBACBA valid
- Making a periodic schedule admissible is always possible, but changes specification…
Admissibility of schedules

- For example: adding initial token changes FIR order
From repetition vector to schedule

- Repeatedly schedule fireable actors up to number of times in repetition vector
  \[ q = \begin{bmatrix} 1 & 2 & 2 \end{bmatrix}^T \]

- Can find either ABCBC or ABBCC

- If deadlock before original state, no valid schedule exists (Lee ‘86)
From schedule to implementation

- Static scheduling used for:
  - behavioral simulation of DF (extremely efficient)
  - code generation for DSP
  - HW synthesis (Cathedral by IMEC, Lager by UCB, ...)

- Issues in code generation
  - execution speed (pipelining, vectorization)
  - code size minimization
  - data memory size minimization (allocation to FIFOs)
  - processor or functional unit allocation
Compilation optimization

- Assumption: *code stitching*  
  (chaining custom code for each actor)
- More efficient than C compiler for DSP
- Comparable to hand-coding in some cases
- Explicit parallelism, no artificial control dependencies
- Main problem: memory and processor/FU allocation depends on scheduling, and vice-versa
Code size minimization

- Assumptions (based on DSP architecture):
  - subroutine calls expensive
  - fixed iteration loops are cheap
    (“zero-overhead loops”)

- Absolute optimum: single appearance schedule (SAS)
  e.g. ABCBC -> A (2BC), ABBCC -> A (2B) (2C)
  - may or may not exist for an SDF graph…
  - buffer minimization relative to single appearance schedules
    (Bhattacharyya ‘94, Lauwereins ‘96, Murthy ‘97)
Assumption: no buffer sharing

Example:

\[ q = \begin{bmatrix} 100 & 100 & 10 & 1 \end{bmatrix}^T \]

- Valid SAS: \((100 \ A) \ (100 \ B) \ (10 \ C) \ D\)
  - requires 210 units of buffer area
- Better (factored) SAS: \((10 \ (10 \ A) \ (10 \ B) \ C) \ D\)
  - requires 30 units of buffer areas, but…
  - requires 21 loop initiations per period (instead of 3)
Dynamic scheduling of DF

- SDF is limited in modeling power
  - no run-time choice
    - cannot implement Gaussian elimination
- More general DF is too powerful
  - non-Static DF is Turing-complete (Buck ‘93)
    - bounded-memory scheduling is not always possible
- BDF: semi-static scheduling of special “patterns”
  - if-then-else
  - repeat-until, do-while
- General case: thread-based dynamic scheduling
  (Parks ‘96: may not terminate, but never fails if feasible)
Example of Boolean DF

- Compute absolute value of average of $n$ samples
Example of general DF

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

- Deterministic merge (no “peeking”)

```
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)
    }
}
```
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)