Data-flow vs Control-flow

• Fuzzy distinction, yet useful for:
  – specification (language, model, ...)
  – synthesis (scheduling, optimization, ...)
  – validation (simulation, formal verification, ...)
• Rough classification:
  – control:
    • don’t know when data arrive (quick reaction)
    • time of arrival often matters more than value
  – data:
    • data arrive in regular streams (samples)
    • value matters most

Data-flow vs Control-flow

• Specification, synthesis & validation methods tend to emphasize ...
  – for control:
    – event/reaction relation
    – response time
    – (Real Time scheduling for deadline satisfaction)
    – priority among events and processes
  – for data:
    – functional dependency between input and output
    – memory/time efficiency
    – (Dataflow scheduling for efficient pipelining)
    – all events and processes are equal

Process Networks

• Communicating processes with directed flow
  – communication: token “stream” between two processes
  – process: operations on tokens
  – host language: process description
  – coordination language: network description

Kahn process networks (1974)

• special class of process networks
• stream is FIFO with unbounded capacity,
• process:
  – destructive read ("consumption") at process start,
  – non-destructive write ("production") at process end,
  – blocking read — process only executed if data available,
  – non-blocking write.
Kahn Process Networks: Formalism

Sequence (a stream) \( X = [x_1, x_2, \ldots] \)
Prefix ordering \([x_1, x_2] \leq [x_1, x_2, x_3]\)
Increasing chain of seq. \( \chi = [x_0, x_1, \ldots] \) where \( X_0 \subseteq X_1 \)
Least upper bound lub \( \chi \subseteq Y \) where \( X_i \subseteq Y \) for all \( X_i \in \chi \)

\[
\begin{array}{c}
\text{FIFO} \\
\text{process} \\
\text{FIFO} \\
\text{process}
\end{array}
\]

\( X \rightarrow F(X) \rightarrow Y \)

Continuous process \( F(\text{lub } \chi) = \text{lub } F(\chi) \)

Kahn Process Networks: Monotonicity

- Monotonicity
  - \( X \subseteq X' \Rightarrow F(X) \subseteq F(X') \)
- It can be proved that...
  - a continuous process is monotone
  - given a part of the input sequence it may be possible to compute part of the output sequence.

Monotonic does not imply continuous

- Consider \( F: S \rightarrow S \)

\[
F(x) = \begin{cases} 
[0] & \text{if } X \text{ is a finite sequence} \\
[0,1] & \text{otherwise}
\end{cases}
\]  

Only two outputs are possible, both finite sequences. To show that this is monotone, note that if the sequence \( X \) is infinite and \( X \subseteq X' \), then \( X = X' \), so

\[
F = F(X) \subseteq F(X') = F'(X')
\]

If \( X \) is finite, then \( F = F(X) = [0] \), which is a prefix of all possible outputs. To show that it is not continuous, consider the increasing chain

\[
\chi = \{X_0, X_1, \ldots\} \text{ where } X_0 \subseteq X_1 \subseteq \ldots
\]

where each \( X_i \) has exactly \( i \) elements in it. Then \( \chi \) is infinite, so

\[
F(\chi) = [0,1] = F(X_1) = [0]
\]

Iterative computation of this function is clearly problematic.

A useful property is that a network of monotonic processes itself defines a monotonic process. This property is valid even for process networks with feedback loops, as is formally proven using induction by Petrenko and Shafir-Zelig [79]. It should not be surprising given the results, so far that one can formally show that networks of monotonic processes are continuous.
Non-monotonic Processes

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

<table>
<thead>
<tr>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>([x_1, x_2, x_3, \ldots])</td>
<td>([x_1, y_1, x_2, y_2, x_3, \ldots])</td>
</tr>
<tr>
<td>([y_1, y_2, y_3, \ldots])</td>
<td>([x_1, y_1, x_2, y_2, x_3, \ldots])</td>
</tr>
</tbody>
</table>

• **Fairness:** every non-empty sequence is processed.

In the previous example, we have:

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

- but

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

• are incomparable

⇒ The process is not monotonic (needs prediction of the future to be really fair).

⇒ The process is not continuous.

In fact the process is not even a (deterministic) function.

Fair Merge

- **Fair merge:**

  - interleave input streams \(X_1\) and \(X_2\) to produce output stream \(Y\).

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 (LFP).

If \(F\) is continuous then the least fixed point exists

\[ LFP = \text{LUB}(\{ F^n(\bot, I) : n \geq 0 \}) \]