HW/SW Codesign

Exercise 2:
Kahn Process Networks and Synchronous Data Flows

1. October 2014

Mirela Botezatu
bmirela@student.ethz.ch
Kahn Process Network (KPN)

• Specification model
  – Proposed as language for parallel programming
  – Processes communicate via First-In-First-Out (FIFO) queues of infinite size
  – **Read**: destructive and blocking
    • A process stays blocked on a *wait* until something is being sent on the channel by another process
  – **Write**: non-blocking
    • A process can never be prevented from performing a *send* on a channel
KPNs: Graphical Representation

- Oriented graph with labeled nodes and edges
  - Nodes: processes
  - Edges: channels (one-directional)
    - Incoming edges with only end vertices: inputs
    - Outgoing edges with only origin vertices: outputs
KPNs: Assumptions and Restrictions

- Processes can communicate *only* via FIFO queues
- A channel transmits information within an unpredictable but *finite* amount of time
- At any time, a process is either computing or waiting on *exactly one* of its input channels
  - *(i.e., no two processes are allowed to send data on the same channel)*
- Each process follows a sequential program
KPNs: Monotonicity

A monotonic process \( F \) generates from an ordered set of input sequences \( X \subseteq X' \) an ordered set of output sequences: \( X \subseteq X' \Rightarrow F(X) \subseteq F(X') \)

- Ordered set of sequences \( X \subseteq X' \) if for each sequence \( i : X_i \subseteq X_i' \)
  
  \([x_1] \subseteq [x_1, x_2] \subseteq [x_1, x_2, x_3, ...]\)

- Explanation:
  - Receiving more input at a process can only provoke it to send more output
  - A process does not need to have all of its input to start computing: future inputs concern only future outputs
KPNs: Determinacy

• A process network is determinate if histories of all channels depend only on histories of input channels
  – History of a channel: sequence of tokens that have been both written and read

• In a determinate process network, functional behavior is independent of timing

• A KPN consisting of monotonic processes is determinate
Adding Non-Determinacy

• Possible ways to introduce non-monotonic behavior
  – Allow processes to perform a non-blocking test for emptiness
  – Allow two or more processes to read from or to write to the same channel
  – Allow processes to share a variable
Synchronous Data Flow (SDF)

• Restriction of KPNs:
  – Allows compile-time scheduling
  – Each process reads/writes a fixed number of tokens at each firing (specified a priori)

• Scheduling in two steps:
  – Establish relative execution rates for the processes (solve a system of linear equations)
  – Determine the periodic schedule(s)

• The schedule can be repeatedly executed without accumulating tokens in the buffers
Synchronous Data Flow (SDF)

- Topology matrix $M$ for a SDF with $n$ processes
  - A **connected** SDF has a periodic schedule **iff** $M$ has rank $r = n - 1$
    (i.e., $Mq = 0$ has a unique smallest integer solution $q \neq 0$)
  - For an **inconsistent** SDF, $M$ has rank $r = n$
    (i.e., $Mq = 0$ has only the all-zeros solution)
  - For a **disconnected** SDF, $M$ has rank $r < n - 1$
    (i.e., $Mq = 0$ has two- or higher-dimensional solutions)

- Example

  $\begin{bmatrix}
  1 & 0 & -1 \\
  1 & -2 & 0 \\
  0 & 3 & -1
  \end{bmatrix}$

  \[ n = 3, \quad \text{rank}(M) = 3 \]

  \[ \Rightarrow \text{inconsistent SDF: there exists no possible schedule to execute it without an unbounded accumulation of tokens} \]
Exercise 2.1.a: “One Peek Merge”

• Merge process that merges data tokens from input channels \( L1 \) and \( L2 \) into one output channel \( out \)

• Two different algorithms are provided

• Examine determinacy
  – *Is the output sequence determined regardless of the arrival order of the input sequences?*

• Examine fairness
  – *Does the process serve the input sequences without letting them starve, even if they have different lengths?*
Exercise 2.1.a: “One Peek Merge”

for (;;) {
    if (test(L1) & test(L2)) {
        X = read(L1); Y = read(L2); write(out,X); write(out,Y); }
    else if (test(L1) & !test(L2)) {
        X = read(L1); write(out,X); }
    else if (!test(L1) & test(L2)) {
        Y = read(L2); write(out,Y); }
}

L1[X]: returns true when a token X is available at channel L1
L1[∅]: returns true when no tokens are available at channel L1

Check if both channels have a token
Exercise 2.1.a: “One Peek Merge”

L1[X]: returns *the serial number* of the token X available at channel L1.

for (;;) {
    if (test(L1) & test(L2)) {
        s1 = getSerial(L1);
        s2 = getSerial(L2);
        if (s1 == s2) {
            X = read(L1); Y = read(L2);
            write(out, X); write(out, Y);
        } else if (s1 < s2) {
            X = read(L1);
            write(out, X);
        } else if (s1 > s2) {
            Y = read(L2);
            write(out, Y);
        }
    } else if (s1 < s2) {
        X = read(L1);
        write(out, X);
    } else if (s1 > s2) {
        Y = read(L2);
        write(out, Y);
    }
}
Exercise 2.1.b

• Draw a KPN that generates the sequence $n(n+1)/2$
• Use basic processes:
  
a) **Sum of two numbers**: sends to the output channel the sum of the numbers received from the two input channels
  
b) **Product of two numbers**: sends to the output channel the product of the numbers received from the two input channels
  
c) **Duplication of a number**: sends to the two output channels the number received from the input channel
  
d) **Constant generation**: sends to the output channel firstly a constant $i$ and then the number received from the input channel
  
e) **Sink process**: waits infinitely often for a number from the input channel and throw it away
Exercise 2.1.b

• Hints:
  • $f(n) = \frac{n(n+1)}{2} = 0+1+2+3+\ldots+n$
  • Transform it into a recursive expression:
    – $f(0) = 0$
    – $f(n) = n+f(n-1), \quad n \geq 1$
  • Draw the KPN starting from the recursive expression
Exercise 2.2.a

- Two SDF graphs are given:

  - Determine the topological matrices
  - Check their consistency (i.e., compute the rank for $M$)
  - If consistent, determine number of firings for each node required to have a periodic execution
Exercise 2.2.b

- A SDF graph is given:

- Determine the topological matrix
- Check its consistency (i.e., compute the rank for $M$)
- If consistent, determine number of firings for each node required to have a periodic execution
Exercise 2.1.a: Solution

- **Non-deterministic:**
  - The output sequence depends on the arrival order of the input sequences

- **Fair:**
  - The two input sequences are served with a *First-Come-First-Serve* policy: the merge process does not let any of them starve

---

**Algorithm 1**

```plaintext
if L1[X], L2[Y] then
    del(X), del(Y), out[X,Y]
else if L1[X], L2[φ] then
    del(X), out[X]
else if L1[φ], L2[Y] then
    del(Y), out[Y]
else if L1[φ], L2[φ] then
    no operation
end if
```
Exercise 2.1.a: Solution

Algorithm 2

if \( L_1[X] = L_2[Y] \) then
  del(X), del(Y), out[X,Y]
else if \( L_1[X] < L_2[Y] \) then
  del(X), out[X]
else if \( L_1[X] > L_2[Y] \) then
  del(Y), out[Y]
end if

- **Deterministic:**
  - The output sequence is determined regardless of the arrival order of the input sequences

- **Unfair:**
  - The merge process lets a longer sequence starve while waiting for a (possibly never appearing) token from the shorter sequence to perform the comparison
Exercise 2.1.b: Solution

- \( f(n) = \frac{n(n+1)}{2} = 0+1+2+3+\ldots+n \)
- \( f(0) = 0, \quad f(n) = n+f(n-1), \quad n \geq 1 \)

Generate \( n=1,2,3,\ldots \)

Compute and store \( f(n) \)
At the beginning:
\( f(0)=0 \) without waiting for \( n \)
Exercise 2.1.b: Solution

• \( f(n) = \frac{n(n+1)}{2} = 0 + 1 + 2 + 3 + \ldots + n \)
• \( f(0) = 0, \quad f(n) = n + f(n-1), \quad n \geq 1 \)

Generate \( n=1,2,3,\ldots \)

Compute and store \( f(n) \)
At the beginning:
\( f(0)=0 \) without waiting for \( n \)
Exercise 2.1.b: Solution

Generate \( n=1,2,3,\ldots \)

Compute and store \( f(n) \)
At the beginning: \( f(0)=0 \) without waiting for \( n \)

\[ x_1, x_2, x_3, x_4, \ldots \]: history of each channel
Exercise 2.2.a: Solution

\[ M = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix} \]

- \( n = 2, \quad \text{rank}(M) = 1 \)
  \( \Rightarrow \) consistent
- Fire numbers: a:1, b:1
- Possible schedules: (BA)*

\[ M = \begin{bmatrix} 2 & -1 \\ 1 & -1 \end{bmatrix} \]

- \( n = 2, \quad \text{rank}(M) = 2 \)
  \( \Rightarrow \) inconsistent
- No schedule can prevent from an unbounded accumulation of tokes
Exercise 2.2.b: Solution

\[ M = \begin{bmatrix}
1 & -1 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & -1 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & -77 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & -1 & 0 \\
0 & 0 & 1 & 0 & 0 & -77 & 0
\end{bmatrix} \]

- \( n = 6, \)
- \( \text{rank}(M) = 5 \)  
  \( (\text{row6=row3+row4+77*row5}) \)  
  \( \Rightarrow \) consistent
- Fire numbers:
  Quelle:77, DCT:77, Q:77, RLC:77, C:1, R:1