习题课

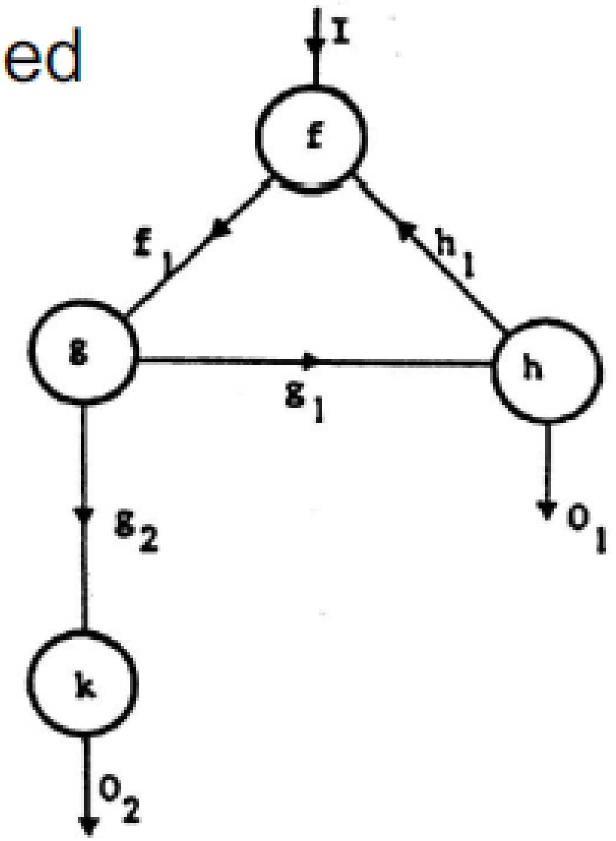
Dr. Kai Huang

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

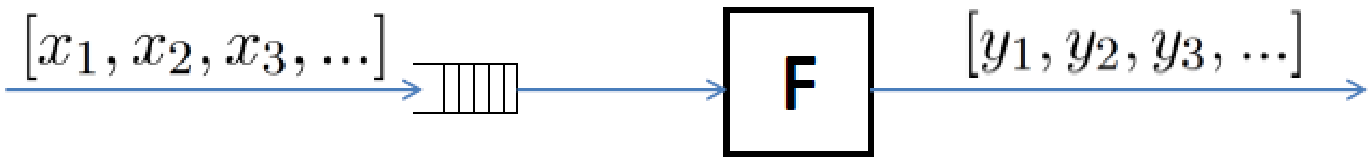
- Oriented graph with labeled nodes and edges
  - Nodes: processes
  - Edges: channels (one-directional)
    - Incoming edges with only end vertices: inputs
    - Outgoing edges with only origin vertices: outputs



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



$$X = [x_1, x_2, x_3, ...]$$

$$[x_1] \subseteq [x_1, x_2] \subseteq [x_1, x_2, x_3, ...]$$

$$\mathbf{X} = (X_0, X_1, ..., X_p) \in S^p$$

$$\mathbf{X} \subseteq \mathbf{X}' \ \ if \ (\forall X_i \subseteq X_i')$$

$$F = S^p \to S^q$$

$$\mathbf{X} \subseteq \mathbf{X}' \Longrightarrow F(\mathbf{X}) \subseteq F(\mathbf{X}')$$

A monotonic process F generates from an ordered set of input sequences X ⊆ X’ to an ordered set of output sequences: X ⊆ X’ ⇨ F(X) ⊆ F(X’)

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**

Synchronous Data Flow (SDF)

Restriction of Kahn Networks to allow compile-time scheduling.

Each process reads and writes a fixed number of tokens each time it fires; firing is an atomic process.

Schedule can be determined completely at compile time (before the system runs).

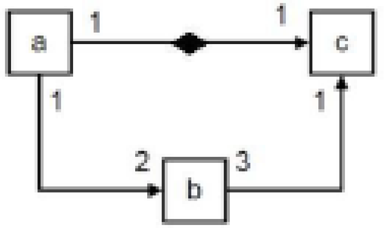
Two steps:

1. Establish relative execution rates by solving a system of linear equations (balancing equations).
2. Determine periodic schedule by simulating system for a single round (returns the number of tokens in each buffer to their initial state).

Result: the schedule can be executed repeatedly without accumulating tokens in 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 $r = n\text{-}1$ (*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 $r < n\text{-}1$ (*i.e.*, $Mq=0$ has two- or higher-dimensional solutions)
- Example



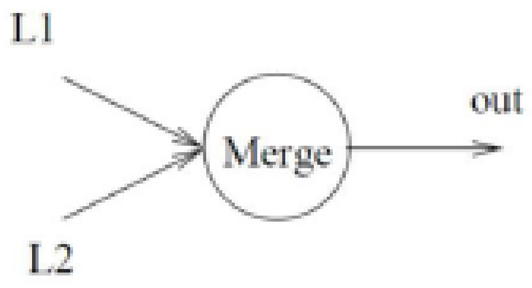
a-c = 0
a-2b =0
3b-c = 0

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

$n = 3$, $rank(M) = 3$
$\Rightarrow$ inconsistent SDF: there exists no possible schedule to execute it without an unbounded accumulation of tokens

Exercise 1.1.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 1.1.a: "One Peek Merge"

Algorithm 1

if L1[X], L2[Y] **then**
  del(X), del(Y), out[X,Y]
else if L1[ϕ], L2[ϕ] **then**

Wait

Check if both channels have a token

Exercise 1.1.a: “One Peek Merge”

Algorithm 2

```
if L1[X] = L2[Y] then
    del(X), del(Y), out[X,Y]
else if L1[X] < L2[Y] then
    del(X), out[X]
else if L1[X] > L2[Y] then
    del(Y), out[Y]
end if
```

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

Check if both channels have a token

```
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); }
  }
}
```

Exercise 1.1.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

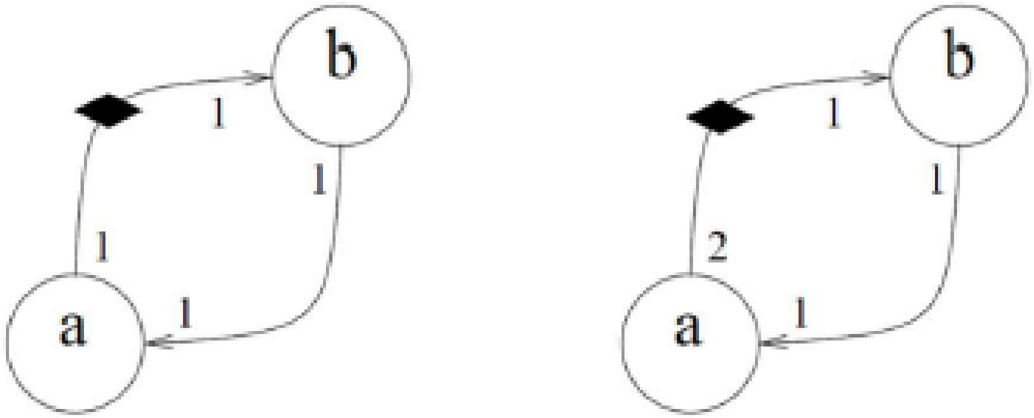
  e) ***Sink process***: waits infinitely often for a number from the input channel and throw it away

Exercise 1.1.b

- **Hints**:
- $f(n) = n(n+1)/2 = 0+1+2+3+\ldots+n$
- Transform it into a recursive expression:
  - $f(0) = 0$
  - $f(n) = n+f(n-1),\ n \geq 1$
- Draw the KPN starting from the recursive expression

Exercise 1.2.a

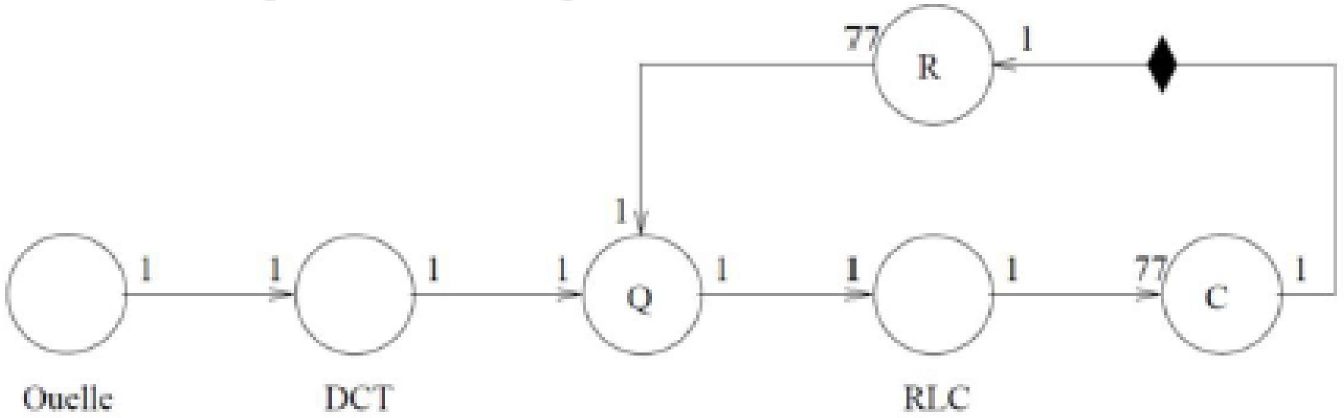
- Two SDF graphs are given:



  - 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 1.2.b

- A SDF graph is given:

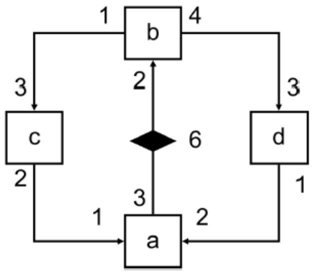


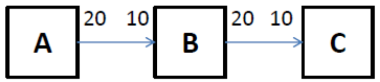
  – Determine the topological matrix
  – Check its consistency (*i.e.*, compute the rank for *M*)
  – If consistent, determine number of firings for each node required to have a periodic execution

Possible schedules:

- (BBBCDDDDAA)*
- (BDBDBCADDA)*
- (BBDDBDDCAA)*
- ...

$$\begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 4 \end{bmatrix}$$





Schedule	Total buffer sizes
(1) ABCBCCC	50 tokens
(2) A(2B)(4 C)	60 tokens
(3) A(2(B (2C)))	40 tokens
(4) A(2(BC))(2 C)	50 tokens