Processes-1
A Formal View

Processes

Flow Graphs

Determinancy

Definitions

- The notion of a sequential process helps to cope with structuring.
- It captures the notion of:
  - Non-determinism
  - Parallel activities
- A process consists of:
  - Program (the text segment)
  - Data (the data segment)
  - A Thread of Execution (i.e., runtime information such as program counter and stack)

Informally, a sequential process or task is the activity resulting from the execution of a program by a sequential CPU

In general: *A process is a program whose execution has started and not yet terminated!*

A simplified view….

- A process may be in any one of several states:
  - Running → the process if using the processor to execute an instruction.
  - Ready → the process is executable but other processes are executing and all CPUs are in use.
  - Blocked → the process awaits an event to occur:
    - Resource availability
    - Messages
    - Signals

- Concurrent processes are individually scheduled to use the CPU.
- Concurrent processing results from multiple instantiations (creation) of a set of processes

$$\boldsymbol{P} = \{p_1, p_2, p_3 \ldots p_n\}$$

- Each process p_i may execute a different program!

A System of Processes

Let $P = \{p_1 \ldots p_n\}$ a set of processes in the system.

We define the following:

- $D(p_i)$ is the domain of p_i
- $R(p_i)$ is the range of p_i

We can view process p_i as a function $p_i = D(p_i) \rightarrow R(p_i)$ that maps memory to memory.

Given two processes, p_i and p_j, we must consider the possibility of interference.

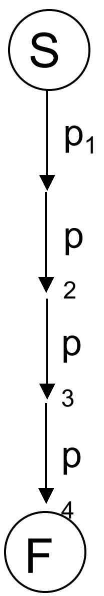
i.e., the order of execution of p_i and p_j may matter!

We can prescribe the execution order of p_i and p_j by a precedence relation "$\rightarrow$".

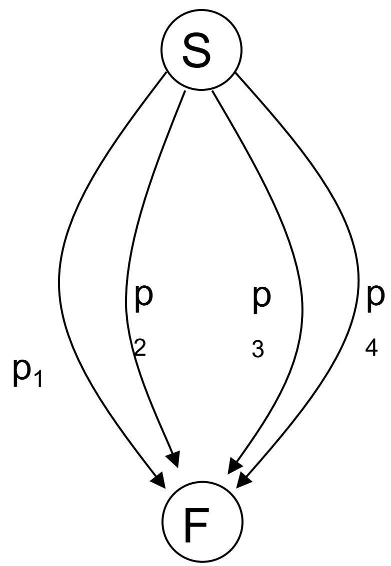
$\rightarrow = \{(p_i,p_j) : p_i$ must complete before the start of $p_j\}$

Process Flow Graph

The precedence constraints among a set of processes can we visualized by a process flow graph.



S(p_1,p_2,p_3,p_4)



P(p_1,p_2,p_3,p_4)

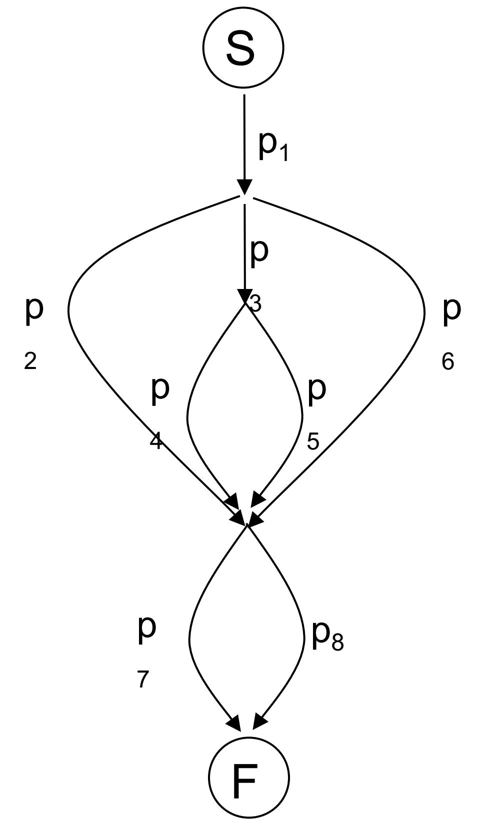
- S and F denote the start and finish of the flow graph, respectively.
- Every PFG is a directed acyclic graph (DAG) [why?]
- $S(p_1,p_2,\ldots,p_n)$ denotes the sequential execution of processes.
- $P(p_1,p_2,\ldots,p_n)$ denotes the parallel execution of processes.

…more flow graph

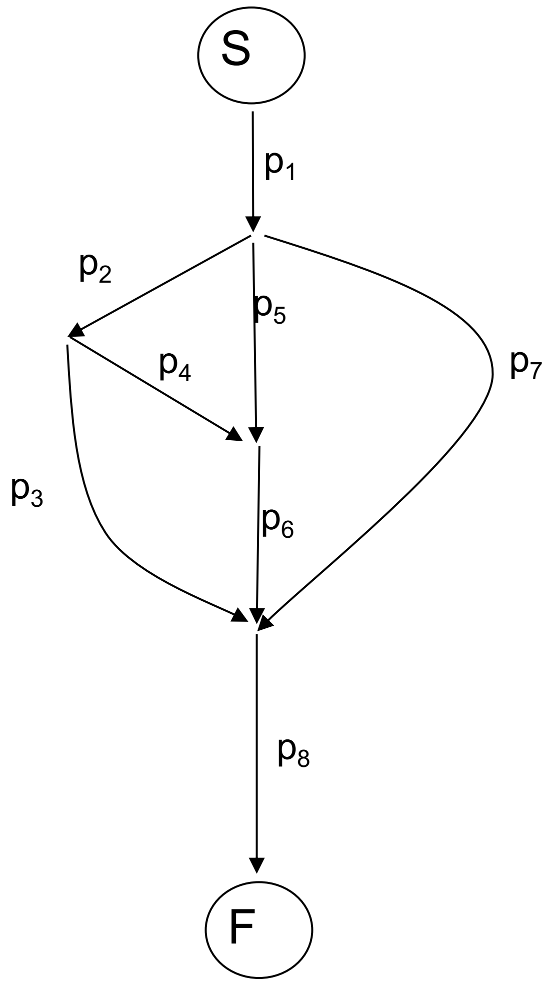
A process flow graph is properly nested if it can be described by the functions S and P and only function composition.

Example:

$S(p_1,P(p_2,S(p_3,P(p_4,p_5)),p_6),P(p_7,p_8))$



…general precedence..

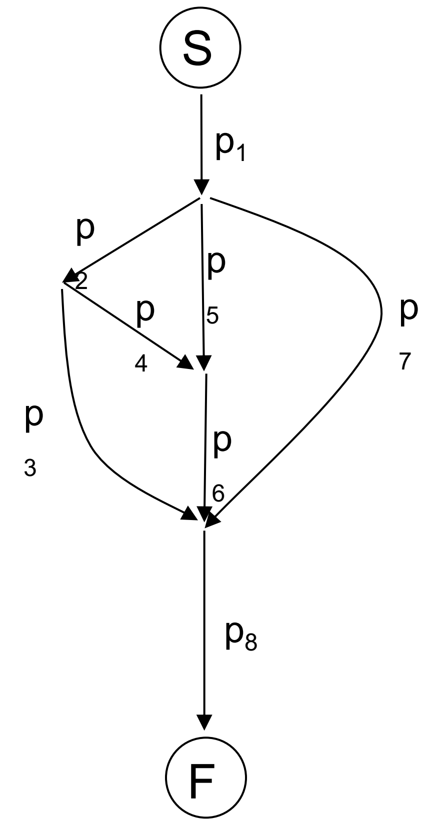


…general precedence..

Note that not all flow graphs are properly nested.

i.e., the precedence relation may not be expressible though S and P operations in conjunction with function composition.

Nevertheless, it is still possible to express the process precedence by using fork() and join() operations. (see example on page 49)



Why is this not properly nested ?

Determinancy

Recall the precedence relation:

$\rightarrow$ = {(p_i,p_j): p_i must complete before the start of p_j}

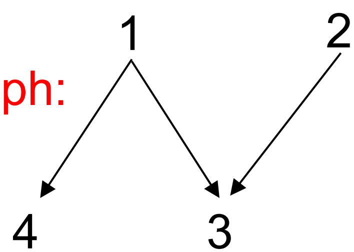
Determinancy: if all executions allowed under "$\rightarrow$" relation result in the same values for all memory cells, then the system is said to be determinate.

Example (non-determinate):

P_1: $C(M_1)+1 \rightarrow M_2$; $C(M_2)-1 \rightarrow M_3$ (concurrent)
P_2: $2^*C(M_2) \rightarrow M_4$
P_3: $C(M_3)+C(M_4) \rightarrow M_5$
P_4: $C(M_2)+1 \rightarrow M_6$

$\rightarrow$ = { (1,3), (2,3),(1,4) }

Precedence Graph:



p_1 and p_2 are independent since (p_1,p_2) is not in $\rightarrow$

Example...

Example (non-determinate):

Initialize: 0 → $M_1..M_6$
P_1: $C(M_1)+1 \rightarrow M_2$; $C(M_2)-1 \rightarrow M_3$ (concurrent)
P_2: $2*C(M_2) \rightarrow M_4$
P_3: $C(M_3)+C(M_4) \rightarrow M_5$
P_4: $C(M_2)+1 \rightarrow M_6$

<u>Conflict</u>
p_1 and p_2 conflict if:

$(D(p_1) \cap R(p_2)) \cup$
$(D(p_2) \cap R(p_1)) \cup$
$(R(p_1) \cap R(p_2)) \neq \emptyset$

	M_1	M_2	M_3	M_4	M_5	M_6
if $p_1 \rightarrow p_2$	0	1	-1	*2*	*1*	2
if $p_2 \rightarrow p_1$	0	1	-1	*0*	*-1*	2

…more determinancy

In summary: A set of processes is determinate, if, given the same input, the same results are produced regardless of the relative speeds of executions of the processes or the legal overlaps in execution.

However, determinancy is not a particularly useful property as it is difficult to determine whether a large system of processes is indeed determinate.

Recall that a process can be viewed as a function:

$$f_p = D(p_i) \rightarrow R(p_i)$$

Supplying f_p is referred to as giving an interpretation.

→→→→ This requires the use of a stronger condition on a system of processes: *mutual non-interference*

mutual non-interference

Two processes p_i and p_j are mutually non-interfering if:

$P_i \rightarrow^* p_j$ or $P_j \rightarrow^* p_i$ or

$(D(p_i) \cap R(p_j)) \cup$
$(D(p_j) \cap R(p_i)) \cup$
$(R(p_i) \cap R(p_j)) = \emptyset$

These conditions are known as Bernstein Conditions!

A system of processes is mutually non-interfering if any two processes meet the above conditions!

- $\rightarrow^*$ means ordered directly or indirectly by $\rightarrow$

Important Theorems

Theorem 1:

A mutually non-interfering (MNI) system of processes is derminate (DET)

MNI → DET

However, the converse is not true: DET ¬→ MNI

Theorem 2:

Consider a system of processes in which, for each process p_i, $D(p_i)$ and $R(p_i)$ are given but the interpretation f_p is left unspecified. If the system is determinate for all interpretations, then all processes are MNI.

PROOF IT !!!

(I mean you in the back row…..)

Maximally Parallel Systems (MPS)

A maximally parallel system of processes is a DET system in which the removal of any (p_i,p_j) from ➔ would cause p_i and p_j to interfere.

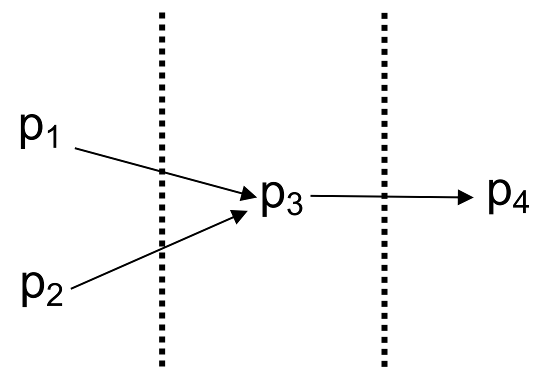
We need to define the transitive closure of ➔:

$cl(➔) = (➔ \cup \{(p_i,p_k) \mid (p_i,p_j)\ \&\ (p_j,p_k)$ forall $p_i,p_j,p_k\ \varepsilon$ ➔ $\})^*$

Example (non max. parallel):

p_1: $M_1 \rightarrow M_2$
p_2: $M_1 \rightarrow M_3$
p_3: $M_2,M_3 \rightarrow M_4$
p_4: $M_2,M_3 \rightarrow M_5$

➔ = {(1,3),(2,3),(3,4)}



Constructing an MPS

Theorem 3:

Given a DET system of processes with precedence relation $\rightarrow$, construct another system with the same processes and new precedence relation $\rightarrow'$ defined by:

$$\rightarrow' = \{(p_i,p_j)\ \varepsilon\ cl(\rightarrow) \mid \underbrace{p_i,p_j \text{ conflict}}_{\text{From Bernstein Conditions}}\}$$

Note: $\rightarrow'$ is unique

In the example:

$\rightarrow = \{(1,3),(2,3),(3,4)\}$

$\rightarrow' = \{(1,3),(1,4),(2,3),(2,4),\cancel{(3,4)}\}$