FINAL EXAM REVIEW

SEG2106 – SOFTWARE CONSTRUCTION

FINAL EXAM TOPICS

Basically, everything we have seen so far:

- Software Development Models
  - Life cycle
  - Black box processes
  - White box processes
  - Waterfall Model
  - Iterative Model
  - Agile Models
- Domain Analysis and Modelling

FINAL EXAM TOPICS

Basically, everything we have seen so far:

- Requirements
  - Client (informal) requirements
  - Functional requirements
    - Use case diagrams
  - Non-functional requirements
- Behavioral Modeling
  - UML Activity diagrams
  - UML State diagrams
  - Petri Nets
  - SDL

FINAL EXAM TOPICS

Basically, everything we have seen so far:

- Lexical Analysis
  - Alphabets
  - Formal languages
    - Operations on formal languages
  - Regular expression
  - Non-deterministic Finite Automata (NFA)
    - Conversion of regular expression to NFA
  - Deterministic Finite Automata (DFA)
    - Implementation of a recognizer
    - Conversion from DFA to NFA
    - Minimization of a DFA

FINAL EXAM TOPICS

Basically, everything we have seen so far:

- Basics of Syntax Analysis
  - Context Free Grammar
  - Derivations
  - Parse trees
  - Grammar ambiguity
  - Top Down Parsing

FINAL EXAM TOPICS

Basically, everything we have seen so far:

- LL(1)
  - Left Recursion (Elimination)
  - Left Factoring
  - FIRST and FOLLOW sets
  - Parsing Tables
  - LL(1) Parsing Algorithm
  - Error Recovery
- Non LL(1) Grammars

FINAL EXAM TOPICS

Basically, everything we have seen so far:

- General Concepts of Concurrency
  - Semaphores
  - Monitors
  - Deadlocks
- Java Concurrency
  - Creating Threads
  - Sleep, Join, Interrupt Operations
  - Java Semaphores
  - Java Intrinsic Locks
  - Java Monitors
  - Atomic Variables

FINAL EXAM TOPICS

Basically, everything we have seen so far:

- Inter-process communication
  - Unicast vs Multicast
  - Asynchronous Message Passing
  - Synchronous Communication
- Introduction to Scheduling
  - First Come First Served (FCFS)
  - Round Robin (RR)
  - Gantt Charts

PART I

MULTIPLE CHOICE QUESTIONS

1)

Given grammar G and language L(G), we can conclude that G is ambiguous when:

A. Using the rules of G, we cannot successfully generate all the strings that belong to L(G)
B. Using the rules of G, we can generate two different parsing trees for a single string in L(G)
C. The grammar is not LL(1) compatible
D. The grammar requires left recursion elimination
E. None of the above

Answer:

B

2)

An LL(1) compatible grammar:

A. Cannot be ambiguous
B. Must be left factored
C. Cannot be left recursive
D. All of the above
E. None of the above

Answer

D

3)

We can conclude that two grammars are equivalent:

A. If their FIRST and FOLLOW sets are the same
B. If they have the same number of terminals and non-terminals
C. If they define the same amount of rules
D. If they generate the same language
E. None of the above

Answer:

D

4)

To use the Java Intrinsic Lock, you must:

A. Create an instance of the Java semaphore
B. Use the ReentrantLock class
C. Create a multi-threaded environment
D. Import the java.util.concurrent package
E. None of the above

Answer:

E

5)

Which one of the following statements is true in regards to inter-process communication (IPC):

A. Communication between processes is always asynchronous
B. Communication between processes is always synchronous
C. Asynchronous communication cannot involve blocking calls
D. Synchronous communication requires the sender to wait until the receiver is ready to receive
E. None of the above

Answer:

D

PART II

DEVELOPMENT QUESTIONS

1) GRAMMAR AMBIGUITY

Prove that the following grammar is ambiguous:

S → xAByz

B → ab | bB | ε

A → a | ε

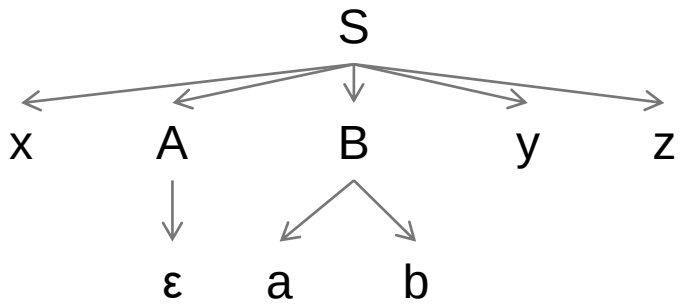
1) GRAMMAR AMBIGUITY - SOLUTION

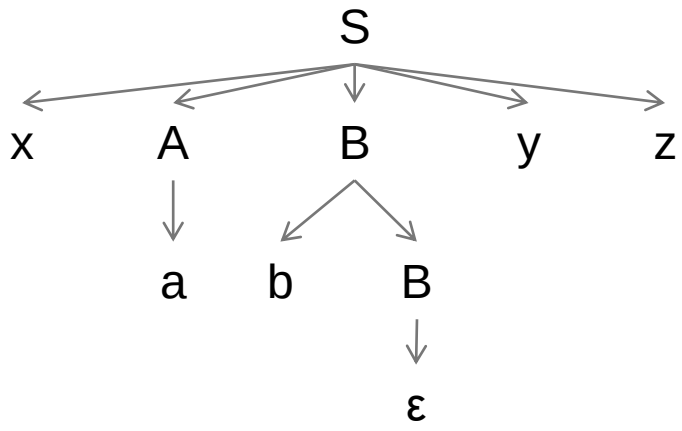
uOttawa
L'Université canadienne
Canada's university

S → xAByz
B → ab | bB | ε
A → a | ε

I just need to identify one sentence that can be generated using two different parse trees.

Here's one: *xabyz*





2) CREATING GRAMMARS + REGULAR EXPRESSIONS

uOttawa
L'Université canadienne
Canada's university

A) Develop a grammar that generates strings that start with an a, followed by two or more b's, followed by zero or more c's and finally followed by several d's. The number of d's should be bigger than the number of b's. Note that a, b, c, and d are all terminal symbols.

B) Create a regular expression that denotes all binary numbers multiple of 2 that represent a decimal number greater than or equal to 8. The binary number can start with leading zeros.

2) CREATING GRAMMARS + REGULAR EXPRESSIONS - SOLUTION

uOttawa
L'Université canadienne
Canada's university

A)

$A \rightarrow abbXdddD$

$C \rightarrow cC \mid \varepsilon$

$D \rightarrow dD \mid \varepsilon$

$X \rightarrow bXd \mid C$

B)

0*1(0|1)*(0|1)(0|1)0

3) LL(1) GRAMMAR

A) Given the following grammar, eliminate left recursion and perform left factoring:

S → SaA | ε

A → bD | bc | bcde | e

D → d | e

3) LL(1) GRAMMAR - SOLUTION

A) Given the following grammar, eliminate left recursion and perform left factoring:

S → SaA | ε

A → bD | bc | bcde | e

D → d | e

Rule for eliminating left recursion:

$$\langle \text{foo} \rangle ::= \langle \text{foo} \rangle \alpha \mid \beta \quad \Rightarrow \quad \langle \text{foo} \rangle ::= \beta \langle \text{bar} \rangle \qquad \langle \text{bar} \rangle ::= \alpha \langle \text{bar} \rangle \mid \varepsilon$$

3) LL(1) GRAMMAR - SOLUTION

uOttawa
L'Université canadienne
Canada's university

A) Given the following grammar, eliminate left recursion and perform left factoring:

S → SaA | ε

A → bD | bc | bcde | e

D → d | e

Rule for left factoring:

$$<A> ::= \alpha\beta_1 | \ \alpha\beta_2 | \ \alpha\beta_3 | \ \dots \ | \ \alpha\beta_n$$

$$<A> ::= \alpha \ <A'>$$
$$<A'> ::= \beta_1 | \ \beta_2 | \ \beta_3 | \ \dots \ | \ \beta_n$$

3) LL(1) GRAMMAR - SOLUTION

uOttawa
L'Université canadienne
Canada's university

A) Given the following grammar, eliminate left recursion and perform left factoring:

S → SaA | ε

A → bD | bc | bcde | e

D → d | e

Eliminating Left Recursion:

S → S′

S′ → aAS′ | ε

3) LL(1) GRAMMAR - SOLUTION

uOttawa
L'Université canadienne
Canada's university

A) Given the following grammar, eliminate left recursion and perform left factoring:

S → SaA | ε

A → bD | bc | bcde | e

D → d | e

Left Factoring:

A → bA′ | e

A′ → D | c | cde

We can further left factor A'

A′ → D | cA′′

A′′ → ε | de

3) LL(1) GRAMMAR - SOLUTION

uOttawa
L'Université canadienne
Canada's university

A) Given the following grammar, eliminate left recursion and perform left factoring:

S → SaA | ε

A → bD | bc | bcde | e

D → d | e

Final grammar:

S → S′

S′ → aAS′ | ε

A → bA′ | e

A′ → D | cA′′

A′′ → ε | de

D → d | e

4) FIRST AND FOLLOW SETS

uOttawa
L'Université canadienne
Canada's university

A) Given the following context free grammar, find the FIRST and FOLLOW sets for each nonterminal.

Note that the terminal symbols of this grammar are:

( ) id num < <= > >= == !=

Therefore, <= is considered a single terminal symbol, not a concatenation of < and =. The same applies to >=, ==, and !=.

1	<compare_expr>::=	(<compare_expr>) <compare_expr'>
2	<compare_expr>::=	id <compare_expr'>
3	<compare_expr>::=	num < compare_expr'>
4	<compare_expr'>::=	<compare_op> <compare_expr> <compare_expr'>
5	<compare_expr'>::=	ε
6	<compare_op>::=	<
7	<compare_op>::=	<=
8	<compare_op>::=	>
9	<compare_op>::=	>=
10	<compare_op>::=	==
11	<compare_op>::=	!=

4) FIRST AND FOLLOW SETS

B) Fill the LL(1) parsing table associated with the above given grammar (by writing the rule number rather than the rule itself in the table cells).

	(	)	id	num	<	<=	>	>=	==	!=	$
compare_expr											
compare_expr'											
compare_op											

C) Based on your parsing table, can you conclude that the grammar is LL(1)? Justify your answer.

4) FIRST AND FOLLOW SETS - SOLUTION

uOttawa
L'Université canadienne
Canada's university

A) Given the following grammar, find the FIRST and FOLLOW sets for each non-terminal.

1	<compare_expr>::=	(<compare_expr>) <compare_expr'>
2	<compare_expr>::=	id <compare_expr'>
3	<compare_expr>::=	num < compare_expr'>
4	<compare_expr'>::=	<compare_op> <compare_expr> <compare_expr'>
5	<compare_expr'>::=	ε
6	<compare_op>::=	<
7	<compare_op>::=	<=
8	<compare_op>::=	>
9	<compare_op>::=	>=
10	<compare_op>::=	==
11	<compare_op>::=	!=

First(compare_expr) = {(,id,num}
First(compare_expr')= {<,<=,>,>=,==,!=,ε}
First(compare_op)= {<,<=,>,>=,==,!=}

1) FIRST(terminal) is {terminal}

2) If A→aα, and a is a terminal:
{a}∈FIRST(A)

3) If A→Bα, and rule B→ε does **NOT** exist:
FIRST(B)∈FIRST(A)

4) If A→Bα, and rule B→ε **DOES** exist:
{(FIRST(B) − ε)∪FIRST(α)}∈FIRST(A)

4) FIRST AND FOLLOW SETS - SOLUTION

uOttawa
L'Université canadienne
Canada's university

A) Given the following grammar, find the FIRST and FOLLOW sets for each non-terminal.

1	<compare_expr>::=	(<compare_expr>) <compare_expr'>
2	<compare_expr>::=	id <compare_expr'>
3	<compare_expr>::=	num < compare_expr'>
4	<compare_expr'>::=	<compare_op> <compare_expr> <compare_expr'>
5	<compare_expr'>::=	ε
6	<compare_op>::=	<
7	<compare_op>::=	<=
8	<compare_op>::=	>
9	<compare_op>::=	>=
10	<compare_op>::=	==
11	<compare_op>::=	!=

Follow(compare_expr) = {$,),<,<=,>,>=,==,!=}
Follow (compare_expr')= {$,),<,<=,>,>=,==,!=}
Follow (compare_op)= {(,id,num }

1) $\{\$\} \in FOLLOW(S)$

2) If A→ αB:
$FOLLOW(A) \in FOLLOW(B)$

3) If A→αBγ, and rule γ→ε does **NOT** exist:
$FIRST(\gamma) \in FOLLOW(B)$

4) If A→αBγ, and rule γ→ε **DOES** exist:
$\{(FIRST(\gamma)-\varepsilon) \cup FOLLOW(A)\} \in FOLLOW(B)$

4) FIRST AND FOLLOW SETS - SOLUTION

uOttawa
L'Université canadienne
Canada's university

B) Fill the LL(1) parsing table associated with the above given grammar (by writing the rule number rather than the rule itself in the table cells).

First(compare_expr) = {(,id,num}
First(compare_expr')= {<,<=,>,>=,==,!=,ε}
First(compare_op)= {<,<=,>,>=,==,!=}

Follow(compare_expr) = {$,),<,<=,>,>=,==,!=}
Follow (compare_expr')= {$,),<,<=,>,>=,==,!=}
Follow (compare_op)= {(,id,num }

1	<compare_expr>::=	(<compare_expr>) <compare_expr'>
2	<compare_expr>::=	id <compare_expr'>
3	<compare_expr>::=	num < compare_expr'>
4	<compare_expr'>::=	<compare_op> <compare_expr> <compare_expr'>
5	<compare_expr'>::=	ε
6	<compare_op>::=	<
7	<compare_op>::=	<=
8	<compare_op>::=	>
9	<compare_op>::=	>=
10	<compare_op>::=	==
11	<compare_op>::=	!=

	(	)	id	num	<	<=	>	>=	==	!=	$
compare_expr											
compare_expr'											
compare_op											

4) FIRST AND FOLLOW SETS - SOLUTION

uOttawa
L'Université canadienne
Canada's university

B) Fill the LL(1) parsing table associated with the above given grammar (by writing the rule number rather than the rule itself in the table cells).

First(compare_expr) = {(,id,num}
First(compare_expr')= {<,<=,>,>=,==,!=,ε}
First(compare_op)= {<,<=,>,>=,==,!=}

Follow(compare_expr) = {$,),<,<=,>,>=,==,!=}
Follow (compare_expr')= {$,),<,<=,>,>=,==,!=}
Follow (compare_op)= {(,id,num }

1	<compare_expr>::=	(<compare_expr>) <compare_expr'>
2	<compare_expr>::=	id <compare_expr'>
3	<compare_expr>::=	num < compare_expr'>
4	<compare_expr'>::=	<compare_op> <compare_expr> <compare_expr'>
5	<compare_expr'>::=	ε
6	<compare_op>::=	<
7	<compare_op>::=	<=
8	<compare_op>::=	>
9	<compare_op>::=	>=
10	<compare_op>::=	==
11	<compare_op>::=	!=

	(	)	id	num	<	<=	>	>=	==	!=	$
compare_expr	*1*		*2*	*3*							
compare_expr'											
compare_op											

4) FIRST AND FOLLOW SETS - SOLUTION

uOttawa
L'Université canadienne
Canada's university

B) Fill the LL(1) parsing table associated with the above given grammar (by writing the rule number rather than the rule itself in the table cells).

First(compare_expr) = {(,id,num}
First(compare_expr')= {<,<=,>,>=,==,!=,ε}
First(compare_op)= {<,<=,>,>=,==,!=}

Follow(compare_expr) = {$,),<,<=,>,>=,==,!=}
Follow (compare_expr')= {$,),<,<=,>,>=,==,!=}
Follow (compare_op)= {(,id,num }

1	<compare_expr>::=	(<compare_expr>) <compare_expr'>
2	<compare_expr>::=	id <compare_expr'>
3	<compare_expr>::=	num < compare_expr'>
4	<compare_expr'>::=	<compare_op> <compare_expr> <compare_expr'>
5	<compare_expr'>::=	ε
6	<compare_op>::=	<
7	<compare_op>::=	<=
8	<compare_op>::=	>
9	<compare_op>::=	>=
10	<compare_op>::=	==
11	<compare_op>::=	!=

	(	)	id	num	<	<=	>	>=	==	!=	$
compare_expr	*1*		*2*	*3*							
compare_expr'		*5*			*4,5*	*4,5*	*4,5*	*4,5*	*4,5*	*4,5*	*5*
compare_op											

4) FIRST AND FOLLOW SETS - SOLUTION

uOttawa
L'Université canadienne
Canada's university

B) Fill the LL(1) parsing table associated with the above given grammar (by writing the rule number rather than the rule itself in the table cells).

First(compare_expr) = {(,id,num}
First(compare_expr')= {<,<=,>,>=,==,!=,ε}
First(compare_op)= {<,<=,>,>=,==,!=}

Follow(compare_expr) = {$,),<,<=,>,>=,==,!=}
Follow (compare_expr')= {$,),<,<=,>,>=,==,!=}
Follow (compare_op)= {(,id,num }

1	<compare_expr>::=	(<compare_expr>) <compare_expr'>
2	<compare_expr>::=	id <compare_expr'>
3	<compare_expr>::=	num < compare_expr'>
4	<compare_expr'>::=	<compare_op> <compare_expr> <compare_expr'>
5	<compare_expr'>::=	ε
6	<compare_op>::=	<
7	<compare_op>::=	<=
8	<compare_op>::=	>
9	<compare_op>::=	>=
10	<compare_op>::=	==
11	<compare_op>::=	!=

	(	)	id	num	<	<=	>	>=	==	!=	$
compare_expr	*1*		*2*	*3*							
compare_expr'		*5*			*4,5*	*4,5*	*4,5*	*4,5*	*4,5*	*4,5*	*5*
compare_op					*6*	*7*	*8*	*9*	*10*	*11*	

4) FIRST AND FOLLOW SETS - SOLUTION

uOttawa
L'Université canadienne
Canada's university

C) Based on your parsing table, can you conclude that the grammar is LL(1)? Justify your answer.

No, this is not an LL(1) grammar since we have multiple productions in one cell of the table.

	(	)	id	num	<	<=	>	>=	==	!=	$
compare_expr	*1*		*2*	*3*							
compare_expr'		*5*			*4,5*	*4,5*	*4,5*	*4,5*	*4,5*	*4,5*	*5*
compare_op					*6*	*7*	*8*	*9*	*10*	*11*	

5) CONCURRENCY

Before you solve this question, you should study the key Semaphore methods described on the next slide. You will use these methods in your solution.

Using semaphores for synchronization, complete the Java code for a Producer/Consumer program that has the following description:

- One producer sends 3 items to the shared buffer at a time
- Two consumers, each removes 2 items from the shared buffer at a time
- The shared buffer can hold 5 items
- Only one entity (producer or consumer) can access the buffer at a time (mutual exclusion)

To complete the program, add your lines of code whenever you are asked to do so.

5) CONCURRENCY

Method Name	Description
`Semaphore(int counter)`	**This is the constructor**. The integer parameter ***counter*** is used to initialize the semaphore's counter.
`void acquire()`	This is a standard acquire method: acquire **one** permit from the semaphore (by decrementing the counter). This call blocks in case a permit is not available (i.e. counter is equal to zero); it remains blocked until a permit becomes available.
`void acquire(int x)`	Acquire ***x*** permits from the semaphore (where ***x*** is the integer parameter passed to the method). This call blocks in case ***x*** permits are not available; it remains blocked until ***x*** permits become available.
`void release()`	This is a standard release method: release **one** semaphore permit (by incrementing the counter).
`void acquire(int y)`	Releases ***y*** semaphore permits (where ***y*** is the integer parameter passed to the method).

5) CONCURRENCY - SOLUTION

uOttawa
L'Université canadienne
Canada's university

```
public class Consumer extends Thread{
   private boolean running;
   private Buffer buffer;
   private Semaphore fullSemaphore, emptySemaphore, mutex;
   private int id;

   public Consumer (int id, Buffer buffer, Semaphore fullSemaphore,
                    Semaphore emptySemaphore, Semaphore mutex){
      running = true;
      this.id = id;
      this.buffer = buffer;
      this.fullSemaphore = fullSemaphore;
      this.emptySemaphore = emptySemaphore;
      this.mutex = mutex;
   }

   private void consume() throws InterruptedException{

      // add code here
      // add code here
      int item1 = buffer.fetch();
      int item2 = buffer.fetch();
      // add code here
      // add code here

   }
```

```
   public void stopRunning(){
      running = false;
   }

   public void run(){
      while (running){
      try{
         // generate a random number between 100 to 500ms
         long time = 100 + (long)(Math.random()*400);
         Thread.sleep(time); // sweet dreams consumer:)

         consume();
      }
      catch (InterruptedException e){
         running = false;
      }
   }
}
```

5) CONCURRENCY - SOLUTION

uOttawa
L'Université canadienne
Canada's university

```
public class Consumer extends Thread{
   private boolean running;
   private Buffer buffer;
   private Semaphore fullSemaphore, emptySemaphore, mutex;
   private int id;

   public Consumer (int id, Buffer buffer, Semaphore fullSemaphore,
                    Semaphore emptySemaphore, Semaphore mutex){
      running = true;
      this.id = id;
      this.buffer = buffer;
      this.fullSemaphore = fullSemaphore;
      this.emptySemaphore = emptySemaphore;
      this.mutex = mutex;
   }

   private void consume() throws InterruptedException{

      fullSemaphore.acquire(2);
      mutex.acquire();
      int item1 = buffer.fetch();
      int item2 = buffer.fetch();
      mutex.release();
      emptySemaphore.release(2);

   }
```

```
   public void stopRunning(){
      running = false;
   }

   public void run(){
      while (running){
      try{
         // generate a random number between 100 to 500ms
         long time = 100 + (long)(Math.random()*400);
         Thread.sleep(time); // sweet dreams consumer:)

         consume();
      }
      catch (InterruptedException e){
         running = false;
      }
   }
}
```

5) CONCURRENCY - SOLUTION

uOttawa
L'Université canadienne
Canada's university

```
public class Producer extends Thread{
   private boolean running;
   private Buffer buffer;
   private Semaphore fullSemaphore, emptySemaphore, mutex;

   public Producer (Buffer buffer, Semaphore fullSemaphore,
                    Semaphore emptySemaphore, Semaphore mutex){
      running = true;
      this.buffer = buffer;
      this.fullSemaphore = fullSemaphore;
      this.emptySemaphore = emptySemaphore;
      this.mutex = mutex;
   }

   public void run(){
      while (running){
      try{
         // generate a random number between 100 to 500ms
         long time = 100 + (long)(Math.random()*400);
         Thread.sleep(time); // sweet dreams producer:)

         produce();
      }
      catch (InterruptedException e){
         running = false;
      }
   }

   public void stopRunning(){
      running = false;
   }

   private void produce() throws InterruptedException{
      // generate three items to store in shared buffer
      int item1 = (int)(Math.random()*10);
      int item2 = (int)(Math.random()*10);
      int item3 = (int)(Math.random()*10);.


      // add code here
      // add code here

      buffer.deposit(item1);
      buffer.deposit(item2);
      buffer.deposit(item3);

      // add code here
      // add code here

   }
}
```

5) CONCURRENCY - SOLUTION

uOttawa
L'Université canadienne
Canada's university

```
public class Consumer extends Thread{
   private boolean running;
   private Buffer buffer;
   private Semaphore fullSemaphore, emptySemaphore, mutex;

   public Producer (Buffer buffer, Semaphore fullSemaphore,
                    Semaphore emptySemaphore, Semaphore mutex){
      running = true;
      this.buffer = buffer;
      this.fullSemaphore = fullSemaphore;
      this.emptySemaphore = emptySemaphore;
      this.mutex = mutex;
   }

   public void run(){
      while (running){
      try{
         // generate a random number between 100 to 500ms
         long time = 100 + (long)(Math.random()*400);
         Thread.sleep(time); // sweet dreams producer:)

         produce();
      }
      catch (InterruptedException e){
         running = false;
      }
   }
```

```
   public void stopRunning(){
      running = false;
   }

   private void produce() throws InterruptedException{
      // generate three items to store in shared buffer
      int item1 = (int)(Math.random()*10);
      int item2 = (int)(Math.random()*10);
      int item3 = (int)(Math.random()*10);.


      emptySemaphore.acquire(3);
      mutex.acquire();

      buffer.deposit(item1);
      buffer.deposit(item2);
      buffer.deposit(item3);

      mutex.release();
      fullSemaphore.release(3);

   }
}
```

5) CONCURRENCY - SOLUTION


```
public class Main {

   public static void main (String [] args){

      int size = 5;

      // create semaphores
      // add code here
      // add code here
      // add code here

      Buffer buffer = new Buffer(size);

      Producer p = new Producer(buffer, fullSemaphore, emptySemaphore, mutex);
      Consumer c1 = new Consumer(1,buffer, fullSemaphore, emptySemaphore, mutex);
      Consumer c2 = new Consumer(2,buffer, fullSemaphore, emptySemaphore, mutex);


      p.start();
      c1.start();
      c2.start();


   }

}
```

5) CONCURRENCY - SOLUTION


```
public class Main {

   public static void main (String [] args){

      int size = 5;

      // create semaphores
      Semaphore fullSemaphore = new Semaphore (0);
      Semaphore emptySemaphore = new Semaphore (size);
      Semaphore mutex = new Semaphore (1);

      Buffer buffer = new Buffer(size);

      Producer p = new Producer(buffer, fullSemaphore, emptySemaphore, mutex);
      Consumer c1 = new Consumer(1,buffer, fullSemaphore, emptySemaphore, mutex);
      Consumer c2 = new Consumer(2,buffer, fullSemaphore, emptySemaphore, mutex);


      p.start();
      c1.start();
      c2.start();


   }

}
```

6) SCHEDULING

Four processes arrive in the order P1, P2, P3, P4.

- P1 burst time: 36
- P2 burst time: 23
- P3 burst time: 3
- P4 burst time: 8

A) Draw the Gantt chart that shows the order in which the scheduler places the processes on the CPU using the Round Robin algorithm given a time quantum of 10 units.

B) Calculate the average waiting time and average completion time for each process.

6) SCHEDULING - SOLUTION

Four processes arrive in the order P1, P2, P3, P4.

- P1 burst time: 36
- P2 burst time: 23
- P3 burst time: 3
- P4 burst time: 8

For Q = 10

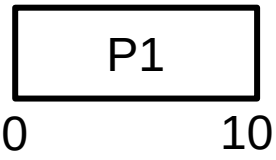
6) SCHEDULING - SOLUTION

uOttawa
L'Université canadienne
Canada's university

Four processes arrive in the order P1, P2, P3, P4.

- P1 burst time: ~~36~~ 26
- P2 burst time: 23
- P3 burst time: 3
- P4 burst time: 8

For Q = 10



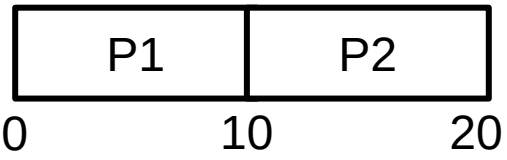
6) SCHEDULING - SOLUTION

uOttawa
L'Université canadienne
Canada's university

Four processes arrive in the order P1, P2, P3, P4.

- P1 burst time: ~~36~~ 26
- P2 burst time: ~~23~~ 13
- P3 burst time: 3
- P4 burst time: 8

For Q = 10



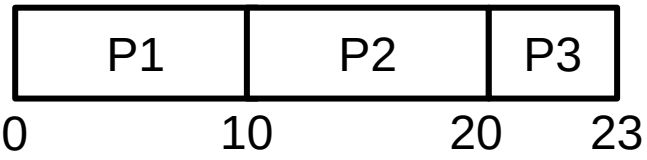
6) SCHEDULING - SOLUTION

uOttawa
L'Université canadienne
Canada's university

Four processes arrive in the order P1, P2, P3, P4.

- P1 burst time: ~~36~~ 26
- P2 burst time: ~~23~~ 13
- P3 burst time: ~~3~~ 0
- P4 burst time: 8

For Q = 10



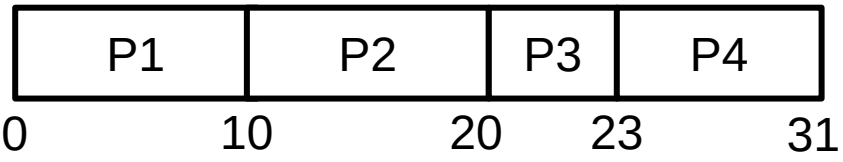
6) SCHEDULING - SOLUTION

uOttawa
L'Université canadienne
Canada's university

Four processes arrive in the order P1, P2, P3, P4.

- P1 burst time: ~~36~~ 26
- P2 burst time: ~~23~~ 13
- P3 burst time: ~~3~~ 0
- P4 burst time: ~~8~~ 0

For Q = 10



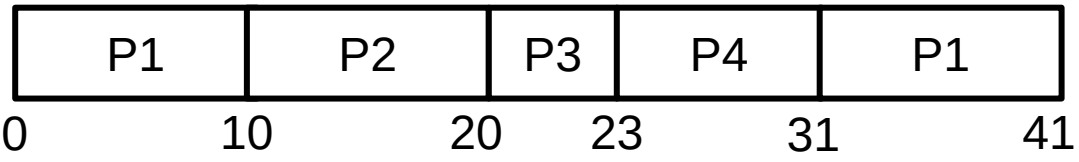
6) SCHEDULING - SOLUTION

uOttawa
L'Université canadienne
Canada's university

Four processes arrive in the order P1, P2, P3, P4.

- P1 burst time: ~~36~~ ~~26~~ 16
- P2 burst time: ~~23~~ 13
- P3 burst time: ~~3~~ 0
- P4 burst time: ~~8~~ 0

For Q = 10



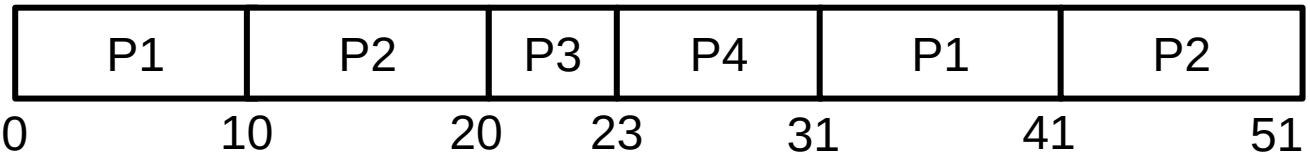
6) SCHEDULING - SOLUTION

uOttawa
L'Université canadienne
Canada's university

Four processes arrive in the order P1, P2, P3, P4.

- P1 burst time: ~~36~~ ~~26~~ 16
- P2 burst time: ~~23~~ ~~13~~ 3
- P3 burst time: ~~3~~ 0
- P4 burst time: ~~8~~ 0

For Q = 10



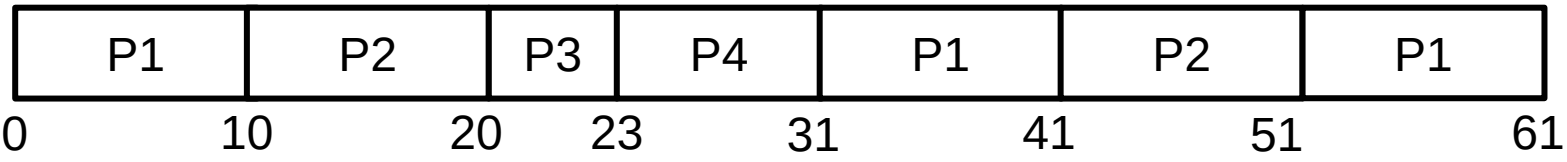
6) SCHEDULING - SOLUTION

uOttawa
L'Université canadienne
Canada's university

Four processes arrive in the order P1, P2, P3, P4.

- P1 burst time: ~~36~~ ~~26~~ ~~16~~ 6
- P2 burst time: ~~23~~ ~~13~~ 3
- P3 burst time: ~~3~~ 0
- P4 burst time: ~~8~~ 0

For Q = 10



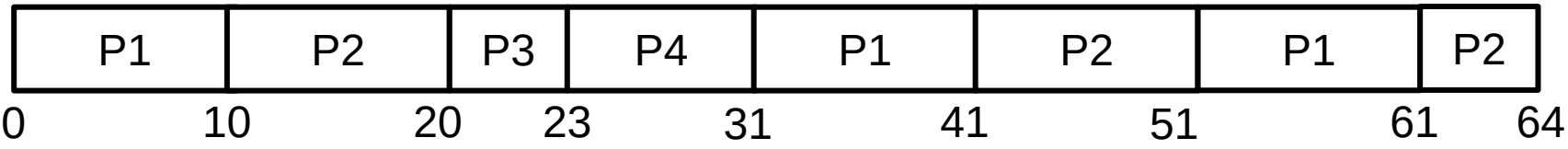
6) SCHEDULING - SOLUTION

uOttawa
L'Université canadienne
Canada's university

Four processes arrive in the order P1, P2, P3, P4.

- P1 burst time: ~~36~~ ~~26~~ ~~16~~ 6
- P2 burst time: ~~23~~ ~~13~~ ~~3~~ 0
- P3 burst time: ~~3~~ 0
- P4 burst time: ~~8~~ 0

For Q = 10



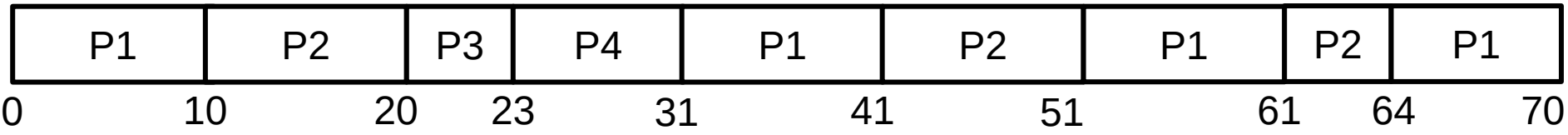
6) SCHEDULING - SOLUTION

uOttawa
L'Université canadienne
Canada's university

Four processes arrive in the order P1, P2, P3, P4.

- P1 burst time: ~~36~~ ~~26~~ ~~16~~ ~~6~~ 0
- P2 burst time: ~~23~~ ~~13~~ ~~3~~ 0
- P3 burst time: ~~3~~ 0
- P4 burst time: ~~8~~ 0

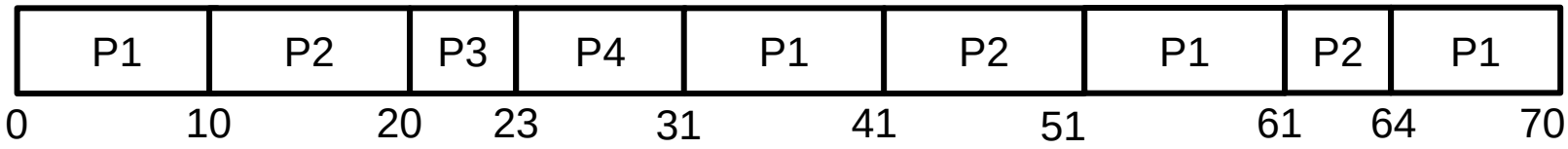
For Q = 10



6) SCHEDULING - SOLUTION

uOttawa
L'Université canadienne
Canada's university

B) Calculate the average waiting time and average completion time for each process.



Wait time:

P1 : (31–10)+(51–41)+(64-61) = 34

P2: (10-0)+(41-20)+(61-51) = 41

P3: (20-0) = 20

P4: (23-0) = 23

Completion Time:

P1 : 70

P2: 64

P3: 23

P4: 31

Average wait time = (34+41+20+23)/4 = 29.5

Average completion time = (70+64+23+31)/4 = 47

GOOD LUCK!

AND DON'T PANIC!