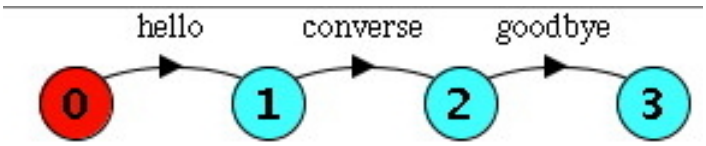
UNIVERSITÉ NICE SOPHIA ANTIPOLIS **POLYTECH'NICE SOPHIA**

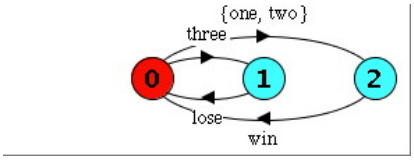
Des questions directement liées au cours

Question 1 ♣ Quel est le processus FSP correspondant à la description LTS suivante (MEETING) :



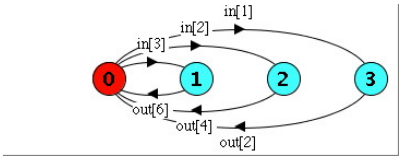
- ☐ `MEETING=(hello -> converse -> goodbye -> MEETING).`
- ☐ `MEETING=(hello -> converse -> goodbye).`
- ☐ `MEETING=(hello | converse | goodbye).`
- ☐ `MEETING=(hello -> converse -> goodbye -> STOP).`
- ☐ `meeting=(hello -> converse -> goodbye).`
- ☐ `MEETING=(hello -> converse -> goodbye)`
- ☐ `P=(hello -> converse -> goodbye).`
- ☐ `P=(hello -> converse -> goodbye -> STOP).`
- ☐ *Aucune de ces réponses n'est correcte.*

Question 2 ♣ Quel est le processus FSP correspondant à la description LTS suivante (GAME) :



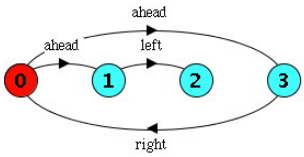
- ☐ `GAME=(one -> P | two -> P | three -> Q), P=(win -> GAME), Q=(lose -> GAME).`
- ☐ `GAME=(one, two -> win -> GAME | three -> lose -> GAME).`
- ☐ `GAME=(one -> win -> GAME | two -> win -> GAME | three -> lose -> GAME).`
- ☐ `GAME=(one -> p | two -> p | three -> r), p=(win -> GAME), q=(lose -> GAME).`
- ☐ `GAME=(one -> P | two -> P | three -> Q). P=(win -> GAME). Q=(lose -> GAME).`
- ☐ *Aucune de ces réponses n'est correcte.*

Question 3 ♣ Quel est le processus FSP correspondant à la description LTS suivante (DOUBLE) :



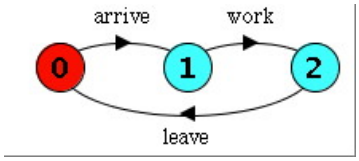
- ☐ `DOUBLE=(in [i:1..3] -> DOUBLE[i]), DOUBLE[j:1..3]=(out [2*J] -> DOUBLE).`
- ☐ `DOUBLE(I=3)=(in [i:1..I] -> out [2*i] -> DOUBLE).`
- ☐ `DOUBLE=(in [i:1..3] -> out [2*i] -> DOUBLE).`
- ☐ *Aucune de ces réponses n'est correcte.*

Question 4 ♣ Quel est le processus FSP correspondant à la description LTS suivante (MOVE) :



- ☐ `MOVE=(ahead -> right -> MOVE | ahead -> left -> STOP).`
- ☐ `MOVE=(ahead -> P | ahead -> left). P=(right -> MOVE).`
- ☐ `MOVE=(ahead -> P | ahead -> Q), P=(right -> MOVE), Q=(left -> STOP).`
- ☐ `MOVE=(ahead -> right -> MOVE | ahead -> left -> MOVE).`
- ☐ *Aucune de ces réponses n'est correcte.*

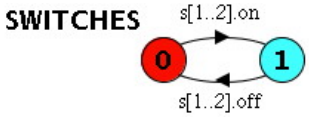
Question 5 ♣ Quel est le processus FSP correspondant à la description LTS suivante (JOB) :

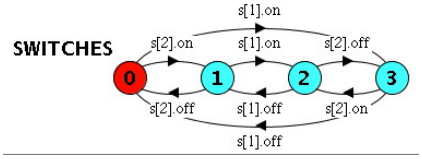


- ☐ `JOB=(arrive -> work -> leave -> job).`
- ☐ `JOB=(arrive -> work -> leave).`
- ☐ `P=(arrive -> work -> leave -> P).`
- ☐ `JOB=(arrive -> work -> leave -> JOB).`
- ☐ `JOB=(arrive | work | leave | JOB).`
- ☐ *Aucune de ces réponses n'est correcte.*

Question 6 Quel est le processus le diagramme LTS correspondant au programme suivant :

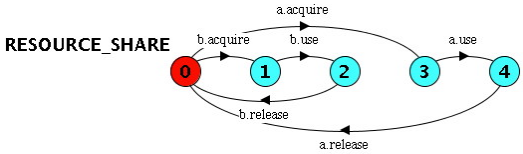
```
SWITCH=(on->off->SWITCH).

||SWITCHES(N=3)=(s[i:1..N]:SWITCH).
```

☐ 

☐ 

Question 7 Quel est le processus le processus LTS correspondant au diagramme FSP suivant :

```
USER=(acquire -> use -> release -> USER).
RESOURCE=(acquire -> release -> RESOURCE).
```



- ☐ `||RESOURCE_SHARE=({a,b}::USER || {a,b}::RESOURCE).`
- ☐ `||RESOURCE_SHARE=({a,b}::USER || {a,b}:RESOURCE).`
- ☐ `||RESOURCE_SHARE=({a,b}:USER || {a,b}:RESOURCE).`
- ☐ `||RESOURCE_SHARE=({a,b}:USER || {a,b}::RESOURCE).`

Question 8 Est-ce que les processus S1 et S2 ont le même comportement ?

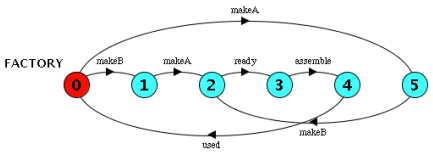
```
P = (a -> b -> P).
Q = (c -> b -> Q).
||S1 = (P || Q).

S2 = (a -> c -> b -> S2 | c -> a -> b -> S2).
```

☐ oui ☐ non

Question 9 Soit le programme FSP suivant :

```
MAKE_A   = (makeA->ready->used->MAKE_A).
MAKE_B   = (makeB->ready->used->MAKE_B).
ASSEMBLE = (ready->assemble->used->ASSEMBLE).

||FACTORY = (MAKE_A || MAKE_B || ASSEMBLE).
```

Est-ce que le diagramme LTS suivant correspond au processus ||FACTORY ?



☐ oui ☐ non

Question 10 Quel est le processus ||S qui permet à plus d'un CLIENT, d'utiliser le SERVEUR ?

```
CLIENT = (appel -> reponse -> traite -> CLIENT).

SERVEUR = (appel-> service -> reponse -> SERVEUR).
```

- ☐ `||S=([1..2]::CLIENT || [1..2]::SERVEUR).`
- ☐ `||S=([1..2]:CLIENT || [1..2]::SERVEUR).`
- ☐ `||S=([1..2]:CLIENT || [1..2]:SERVEUR).`
- ☐ `||S=([1..2]::CLIENT || [1..2]:SERVEUR).`

Question 11 ♣ Quel est le processus ||S qui permet à un CLIENT, d'utiliser le SERVEUR ?

```
CLIENT = (appel -> attend -> traite -> CLIENT).

SERVEUR = (requete -> service -> reponse -> SERVEUR).
```

- ☐ `||S=(CLIENT || SERVEUR)/{requete/appel, reponse/attend}.`
- ☐ `||S=(CLIENT || SERVEUR}.`
- ☐ `||S=(CLIENT || SERVEUR)/{appel/requete, reponse/attend}.`
- ☐ *Aucune de ces réponses n'est correcte.*

Question 12 Quel est le processus Q qui garanti que le processus S accepte une, deux, trois ou quatre actions 'up' avant une action 'down'.

```
P = (up -> down -> P).

||S = (P || Q).
```

- ☐ `Q=(up -> down -> Q).`
- ☐ `Q=(up -> up -> up -> up -> down -> Q).`
- ☐ `Q=(up -> up -> up -> down -> Q).`
- ☐ `Q=(up -> up -> down -> Q).`

Question 13 ♣ En programmation concurrente, à quoi correspond une section critique :

- ☐ Une section critique est une portion de code dans laquelle il doit être garanti qu'il n'y aura jamais plus d'un thread simultanément.
- ☐ Un processus qui exécute une section critique ne peut jamais être interrompu.
- ☐ Une section critique correspond à une succession d'action atomique.
- ☐ Une section critique est une portion de code dans laquelle toute erreur de programmation est interdite.
- ☐ Une section critique est toujours exécutée en exclusion mutuelle.
- ☐ *Aucune de ces réponses n'est correcte.*

Question 14 ♣ En programmation concurrente, à quoi sert un verrou :

- ☐ A contrôler l'usage d'un ressource partagée
- ☐ A proteger une section critique
- ☐ A suspendre l'exécution d'un thread
- ☐ A ouvrir une porte
- ☐ *Aucune de ces réponses n'est correcte.*

Question 15 ♣ Quelle sont les conditions nécessaires pour avoir des données incohérentes :

- ☐ Une ressource exclusive
- ☐ Une seule thread
- ☐ Plusieurs threads
- ☐ Une ressource partagée
- ☐ *Aucune de ces réponses n'est correcte.*

Question 16 Est-ce que le code suivant contrôle correctement l'accès en section critique ?

```
// variable commune aux threads, initialisée à faux
bool occupé = FALSE;

// Chaque thread souhaitant exécuter le code de la section critique exécute le code suivant :
while (occupé) {;}
occupé = TRUE;

// Chaque thread terminant d'exécuter le code de la section critique exécute le code suivant :
occupé = FALSE;
```

☐ oui ☐ non

Question 17 ♣ Quels sont les conditions nécessaire pour provoquer un interblocage (deadlock) :

- ☐ Plusieurs objets partagés protégés par un verrou
- ☐ Un seul objet partagé protégé par un verrou
- ☐ Une seule thread
- ☐ Plusieurs objets partagés non protégés
- ☐ Plusieurs threads
- ☐ *Aucune de ces réponses n'est correcte.*

Question 18 Est-ce que ces deux opérations read et write ci-dessous, regroupées dans la même classe Disk sont correctement programmées ?

```
// Les opérations seek, read et write s'exécutent en exclusion mutuelle
int disk_read (sector x) {
        int r;
        D.seek(x);
        r := D.read();
        return (r);
}

void disk_write (sector x, int v) {
     D.seek(x);
     D.write(v);
}
```

☐ oui ☐ non

Question 19 Est-ce que le code Java suivant implémente bien l'accès à une section critique ?

```
Semaphore mutex = new Semaphore (0);
mutex.Acquire ();
// je suis en section critique
mutex.Release ();
```

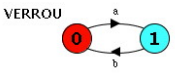
☐ non ☐ oui

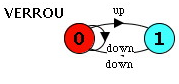
Question 20 Combien de ligne faut-il modifier pour avoir une tranformation FSP -> Java correcte ?

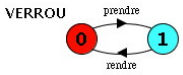
```
BRIDGE = BRIDGE[0],
BRIDGE[n:T] = (when(n==0) enter -> BRIDGE[n+1]
              |exit -> BRIDGE[n-1]).

Class BRIDGE {
  private int n  = 1;
  synchronized void enter() throws xxx {
    while (n!=0) wait();
    n++;
    notifyAll();
  }
  synchronized void exit() {
    // while (false) wait(); -- ligne inutile
    --n;
   notifyAll();
  }
}
```

☐ 0 ☐ 2 ☐ 1 ☐ 4

Question 21 ♣ Quels sont les automates suivants qui représentent le fonctionnement d'un verrou :

☐ 

☐ 

☐ 

☐ *Aucune de ces réponses n'est correcte.*

Question 22 Sachant que le moniteur suivant est utilisé uniquement par 2 threads, est-ce que le programme Java constitue une transformation correcte de la description FSP ?

```
VOIE_UNIQUE = VOIE_UNIQUE[0],
VOIE_UNIQUE[n:T] = (when(n==0) entree_est -> VOIE_UNIQUE[n+1]
              |when(n==0) entree_ouest -> VOIE_UNIQUE[n+1]
              |exit -> VOIE_UNIQUE[n-1]).

Class VOIE_UNIQUE {
  private int n  = 0;
  synchronized void entree() throws xxx {
    // utilisé à la place de entree_est et entree_ouest
    if (n!=0) wait();
    n++;
  }
  synchronized void exit() {
    --n;
   notify();
  }
}
```

☐ non ☐ oui

Question 23 Quel est le code FSP d'un verrou (LOCK) qui a comme alphabet getread, getwrite, release.

- getread : donne le verrou en mode partagé, plusieurs getread sont possibles
- getwrite : donne le verrou en mode exclusif
- release : suit nécessairement une action de verrouillage et remet le verrou dans son état initial (si le verrous était en mode partagé, il faut autant de release que de getread pour le libérer).

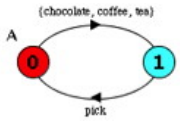
☐
```
VERROU = (getread -> LOCKREAD | getwrite -> LOCKWRITE),
LOCKREAD = (getread -> LOCKREAD
| release -> VERROU),
LOCKWRITE = (release -> VERROU).
```

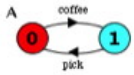
☐
```
VERROU = (getread -> LOCKREAD[1] | getwrite -> LOCKWRITE),
LOCKREAD[i:1..3] = (getread -> LOCKREAD[i+1]
| release -> LOCKREAD[i-1]),
LOCKWRITE = (release -> VERROU).
```

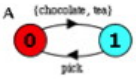
☐
```
VERROU = (getread -> LOCKREAD[1] | getwrite -> LOCKWRITE),
LOCKREAD[i:1..3] = (getread -> LOCKREAD[i+1]
| when (i>1) release -> LOCKREAD[i-1]
| when (i==1) release -> VERROU),
LOCKWRITE = (release -> VERROU).
```

Question 24 Quel est le schéma correct ?

```
NORMAL = (coffee->pick->NORMAL
|tea->pick->NORMAL
|chocolate->pick->NORMAL).
||A=(NORMAL)«{coffee}.
```

☐ 

☐ 

☐ 

Question 25 Comment détecte-t-on sur un diagrame LTS qu'un programme est 'vivant'.

☐0 ☐1 ☐2 ☐4

.....

.....

.....

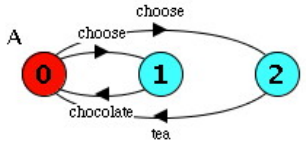
.....

.....

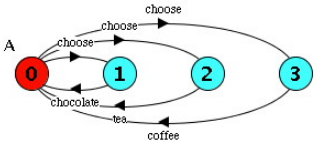
Question 26 Quel est le schéma correct ?

```
ANORMAL = (choose->coffee->ANORMAL
|choose->tea->ANORMAL
|choose->chocolate->ANORMAL).
||A=(ANORMAL)«{coffee}.
```

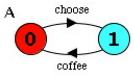
☐



☐



☐



Question 27 Comment détecte-t-on sur un diagrame LTS qu'un programme est sûr.

☐0 ☐1 ☐2 ☐4

.....

.....

.....

.....

.....

Question 28 Donnez la définition de la propriété de sûreté.

☐0 ☐1 ☐2 ☐4

.....

.....

.....

.....

.....

Question 29 Donnez la définition de la propriété de vivacité. ☐0 ☐1 ☐2 ☐4

.....

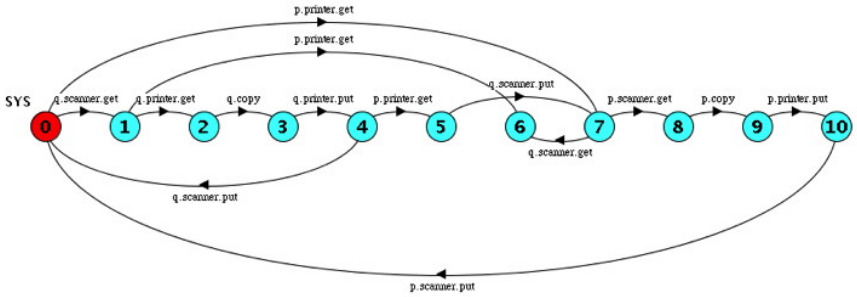
.....

.....

.....

.....

Question 30 Est-ce que le diagramme FSP suivant présente un interblocage ?



☐ non ☐ oui

Question 31 Est-ce que le code suivant présente un risque d'interblocage ?

```
// Initialisation
Semaphore r1 = new Semaphore(1);
Semaphore r2 = new Semaphore(1);

// Processus A
r1.Down();
r2.Down();
use_resources();
r2.Up();
r1.Up();

// Processus B
r2.Down();
r1.Down();
use_resources();
r1.Up();
r1.Up();
```

☐ oui ☐ non

Allocations multiples

- Plusieurs processus se partagent des ressources communes dont le nombre total N est fixé.
- Chaque processus demande un nombre quelconque n de ces ressources (n est variable selon les processus), les utilise, puis les libère.
- Les ressources acquises par un processus sont indisponibles pour les autres jusqu'à ce qu'elles soient libérées (accès exclusif).
- Si le nombre n de ressources demandées par un processus est supérieur au nombre de ressources disponibles (non allouées aux autres processus), sa demande n'est pas satisfaite (aucune ressource ne lui est allouée) et il se bloque. Il sera réveillé et recevra ses ressources quand l'allocation deviendra possible.

Le code de chaque processus consiste en une succession de demande de ressources et se termine, en une ou plusieurs fois par la libération de l'ensemble des ressources réservées.

```
//  Le code de chaque processus consiste en une succession d'opération
// est le suivant~:
< début du programme >
demander (n);

< utilisation des ressources >
libérer (n);
< reste du programme >
```

```
 0 // variables globales partagées
 1 mutex    : sémaphore  init 0;
 2 attente  : sémaphore  init 0;
 3 n_libre  : entier  init N;        // nombre de ressources disponibles
 4 n_att    : entier  init 0;        // nombre de processus en attente

 5 Procédure demander (n) {       14 Procédure  libérer (n) {
 6     mutex.down();              15     mutex.down();
 7     tantque (n>n_libre) {      16     n_libre:=n_libre+n;
 8         n_att := n_att+1;      17     tantque (n_att>0) {
 9         attente.down();        18         attente.up();
 10    }                          19         n_att := n_att-1;
 11    n_libre:=n_libre - n;      20     }
 12    mutex.up();                21     mutex.up();
 13 }                             22 }
```

Question 32 Le programme précédent comportent deux erreurs. Modifiez les deux lignes nécessaire à un fonctionnement correct. ☐0 ☐1 ☐2 ☐4

.....

.....

.....

.....

.....

Question 33 En supposant que le code des procédures demandés et libérés soient correct. Est-ce que le scénario suivant présente un risque d'interblocage ?

- `Nombre de ressources fixés à 20`
- `Demandes successives du processus 1 : 5, 8, 7, 5`
- `Demandes successives du processus 2 : 3, 3, 3`
- `Demandes successives du processus 3 : 4, 4, 3`

☐ non ☐ oui

Question 34 Est-il possible de résoudre le problème de l'allocation successive de ressources par l'algorithme du banquier ?

☐ non ☐ oui

Dans les différents tableaux suivants, quels sont les état sains ?

A	Util	Max
P 1	0	10
P 2	2	9
P 3	4	8
P 4	6	7
Libre : 1		

B	Util	Max
P 1	0	10
P 2	2	9
P 3	4	8
P 4	4	7
Libre : 2		

C	Util	Max
P 1	3	10
P 2	3	9
P 3	3	8
P 4	3	7
Libre : 3		

D	Util	Max
P 1	6	10
P 2	6	9
P 3	3	8
P 4	3	7
Libre : 3		

Question 35 Etat A :

☐ non ☐ oui

Question 36 Etat B :

☐ non ☐ oui

Question 37 Etat C :

☐ oui ☐ non

Question 38 Etat D :

☐ non ☐ oui

Allocations de plusieurs ressources

On dispose d'un ensemble de ressources notées R1 à Rn, manipulées par des processus concurrents à l'aide des procédures suivantes :

- Opération (Ri) effectue un traitement, dont le détail n'est pas fourni, sur la ressource Ri.
- Opération_Composée (Ri, Rj) effectue un traitement portant globalement sur deux ressources distinctes Ri et Rj en exécutant séquentiellement Opération (Ri) et Opération (Rj).

Dans cet exercice on s'intéresse à modifier le code de la procédure Opération_Composée pour garantir la propriété d'atomicité. L'atomicité (propriété du "tout ou rien") requiert que les ressources Ri et Rj soient manipulées globalement en exclusion.

Dans la suite de l'exercice, Mutex désigne un sémaphore d'exclusion mutuelle (i.e. un sémaphore initialisé à 1) et Si, un sémaphore privé associé à la ressource Ri (i.e. un sémaphore initialisé à 0).

Pour chacun des propositions suivantes, nous vous demandons d'étudier les 3 propriétés suivantes :

- Est-ce que la propriété d'atomicité est garantie ?
- Y-a-t-il un risque d'interblocage ?
- Est-il possible d'exécuter en parallèle deux Opération_Composée portant sur des ressouces différentes ?

Proposition 1 : P(Mutex); Opération (Ri); Opération (Rj); V(Mutex);

Question 39

☐ pas atomicité ☐ atomicité

Question 40

☐ pas interblocage ☐ interblocage

Question 41

☐ pas parallélisme ☐ parallélisme

Proposition 2 : P(Si); Opération(Ri); V(Si); P(Sj); Opération(Rj); V(Sj);

Question 42

☐ atomicité ☐ pas atomicité

Question 43

☐ pas interblocage ☐ interblocage

Question 44

☐ pas parallélisme ☐ parallélisme

Proposition 3 : P(Si); P(Sj); Opération(Ri); Opération(Rj); V(Si); V(Sj);

Question 45

☐ pas atomicité ☐ atomicité

Question 46

☐ pas interblocage ☐ interblocage

Question 47

☐ parallélisme ☐ pas parallélisme