Université de Lorraine: Diplôme Telecom Nancy 2A IL
Epreuve: ASPD
Horaires et lieu : 8 h 00 à 9 h 00 le lundi 4 mai 2026
Durée de l'épreuve : 1 h 00
Nom du rédacteur : Dominique Méry
Tous les documents personnels sont autorisés

Epreuve TP



La clarté de la rédaction et la justification des réponses sont des éléments essentiels de l'appréciation. Les exercices sont indépendants. Vous devez répondre à chaque exercice sous la forme d'un dossier dont le nom sera le nom de l'exercice. Ce dossier contiendra les fichiers .tla et .cfg nécessaires pour évaluer leur pertinence par rapport aux questions. Le sujet comprend quatre (4) exercices.

TP



Votre travail sera rendu sous la forme d'une archive .zip qui est constituée d'un dossier principal portant votre nom et de 4 sous-dossiers appelés ex1, ex2, ex3, ex4 et contenant les éléments demandés aux questions respectives. Vous enverrez cette archive à dominique.mery@loria.fr avec le sujet aspd26. Faites attention que l'archive ne soit pas vide.

On rappelle pour les prochains exercices qu'un programme PlusCal peut contenir plusieurs processus et que chaque processus est traduit par une relation NEXT spécifique. Le système global est alors défini par la disjonction des relations Next composant le programme. Ensuite, on traduit automatiquement le programme et ses processus en une suite d'actions et de relations next. Une des relations next est utilisée pour le modèle qui produit toutes les exécutions possibles selon la relation next à partir d'une assertion définissant les conditions initiales.

Exercice 1. *Soit le programme Pluscalde la figure 1. Compléter le module pluscalil267q, en remplaçant A1, A2, B1, B2, A et B, afin que la vérification ne détecte pas d'erreurs. Les expressions A1, A2, B1, B2, A et B sont des ensembles et A et B ne contiennent qu'une seule valeur. De plus, les ensemebles A1, A2, B1, B2, A et B sont minimaux: cela signifie que si on enlève une valeur, alors la vérification échoue et détecte une erreur. Il est clair que la propriété Safe doit aussi être validée avec les contraintes indiquées. Vous pouvez ajouter un fichier explicatif si vous le souhaitez ou ajouter un commentaire dans le fichier TLA$^+$.*

Exercice 2. *Soit le programme Pluscal de la figure 2. Compléter le module pluscalil272q, en remplaçant A1, A2, B1, B2, l, r, afin que la vérification ne détecte pas d'erreurs. Les expressions A1, A2, B1, B2 sont des ensembles . De plus, les ensemebles A1, A2, B1, B2 sont minimaux: cela signifie que si on enlève une valeur, alors la vérification échoue et détecte une erreur.Vous pouvez ajouter un fichier explicatif si vous le souhaitez ou ajouter un commentaire dans le fichier TLA$^+$.*

Exercice 3. *Soit le programme Pluscal des figures 3 et 4. Ce programme calcule une valeur à partir des valeurs de A. Dans le cas présent, A est un tableau affecté des 8 premières valeurs de $\mathbb{N}$ non nulles.*

Compléter ce module en écrivant la propriété SafePC permettant de vérifier que le calcul réparti fonctionne correctement. On ne se préoccupe pas de faire varier A.

Listing 1: pluscalil267q.tla

```
1  ---------------------------- MODULE pluscalil267q ----------------------------
2  EXTENDS Integers, Sequences, TLC, FiniteSets
3
4
5  (*
6  --algorithm ex1{
7  variables x = 1, y = 2;
8
9
10 process (one = 1)
11 {
12   A:
13     x := x + 1;
14   B:
15     y := y -1;
16   C:
17       assert  x \in A1 /\ y \in B1;
18 };
19
20 process (two = 2)
21 {
22   D:
23     x := x - 1;
24   E:
25     y:=y+2;
26    F:
27     x:= x+2;
28
29    G: assert  x \in A2 /\ y \in B2;
30 };
31
32 }
33 end algorithm;
34
35 *)
36
37 Safe ==   (\A self \in ProcSet: pc[self] = "Done") => x \in A /\ y \in B
38
39 ===========================================================================
```

Figure 1: Algorithme du listing 1

Listing 2: pluscalil272q.tla

```
1  ---------------------------- MODULE pluscalil272q ----------------------------
2  EXTENDS Integers, Sequences, TLC, FiniteSets
3  CONSTANTS l,r
4  ASSUME .....
5  (*
6  --algorithm ex3
7  variables i = 0, j = 0,k,halt1=FALSE,halt2=FALSE;
8
9  process one = 1
10 begin
11 wi:while i < 10 do
12           i: assert i \in A1  /\ i+j \in B1;
13           i := i+1;
14 end while;
15 final1: halt1 := TRUE;
16 end process;
17
18 process two = 2
19 begin
20 wj:while i+j  < 10 do
21           j: assert j \in A2 /\ i+j \in B2;
22       j := j+1;
23 end while;
24 final2: halt2 := TRUE;
25   end process;
26
27 process three = 3
28 begin
29 w3:  await halt1=TRUE /\ halt2=TRUE;
30 final3: k := i + j;
31 check: assert k \in l..r
32 end process
33 end algorithm;
34
35 *)
36
37 ===============================================================================
```

Figure 2: Algorithme du listing 2

Listing 3: pluscalil273qa.tla

```
1  ---------------------- MODULE pluscalil273q ----------------------
2  \* CalculParalleleCanaux -----
3  EXTENDS Integers, Sequences, TLC
4  CONSTANTS
5          special \* symbole spÃ©cial de marque de fin
6
7  (*--algorithm  calcul
8
9  variables
10     A = <<1,2,3,4,5,6,7,8>>, \* affectation des donnÃ©es
11     chan1 = <<>>, chan2 = <<>>, chan3 = <<>>, \* canaux  de communication
12     chanR = <<>>,   \* canal  des  rÃ©sultats
13     eod = special,
14      total = 0;
15 \* ----------------== DISTRIBUTEUR ----------------------!
16 process P0 = "Distributeur"
17 variable i = 1;
18 begin
19    w: while i <= Len(A) do
20          if i % 3 = 1 then
21              chan1 := Append(chan1, A[i]);
22          elsif i % 3 = 2 then
23              chan2 := Append(chan2, A[i]);
24          else
25              chan3 := Append(chan3, A[i]);
26          end if;
27          i := i + 1;
28     end while;
29
30     \* fin de donnÃ©es
31     chan1 := Append(chan1, eod);
32     chan2 := Append(chan2, eod);
33     chan3 := Append(chan3, eod);
34 end process;
35
36
37
38 ==========
```

Figure 3: Algorithme du listing 3

Listing 4: pluscalil273qb.tla

```
1  ---------- MODULE pluscalil273q ----------
2  \* suite
3  \* ---------- P1 ---------- !
4  process P1 = "P1"
5  variable x, sum = 0;
6  begin
7      w:while TRUE do
8          await Len(chan1) > 0;
9          x := Head(chan1);
10         chan1 := Tail(chan1);
11
12         if x = eod then goto Send1;
13         else sum := sum + x;
14         end if;
15     end while;
16
17 Send1:
18     chanR := Append(chanR, sum);
19 end process;
20
21 \* ---------- P2 ---------- !
22 process P2 = "P2"
23 variable x, sum = 0;
24 begin
25     w:while TRUE do
26         await Len(chan2) > 0;
27         x := Head(chan2);
28         chan2 := Tail(chan2);
29
30         if x = eod then goto Send2;
31         else sum := sum + x;
32         end if;
33     end while;
34
35 Send2:
36     chanR := Append(chanR, sum);
37 end process;
38
39 \* ---------- P3 ---------- !
40 process P3 = "P3"
41 variable x, sum = 0;
42 begin
43     w:while TRUE do
44         await Len(chan3) > 0;
45         x := Head(chan3);
46         chan3 := Tail(chan3);
47
48         if x = eod then goto Send3;
49         else sum := sum + x;
50         end if;
51     end while;
52
53 Send3:
54     chanR := Append(chanR, sum);
55 end process;
56
57 \* ---------- COMBINAISON ---------- !
58 process Combine = "Combine"
59 variable x, count = 0;
60 begin
61    w: while count < 3 do
62         await Len(chanR) > 0;
63         x := Head(chanR);
64         chanR := Tail(chanR);
65
66         total := total + x;
67         count := count + 1;
68     end while;
69
70     print <<"Somme totale:", total>>;
71 end process;
72
73 end algorithm; *)
74
75
76 SafePC == .......
77
78 ============================
```

Figure 4: Algorithme du listing 4

Exercice 4. *Soit le module pluscalil268q des figures 5 et 6 qui permet de coordonner l'utilisation d'une ressource R partagée par des clients de deux serveurs S1 et S2. S1 sert p clients de type 1 et S2 sert q clients de type 2*

Question 4.1. *Compléter le module pluscalil268.tla, en précisant clairement que p et q sont des entiers naturels non nuls.*

Question 4.2. *Initialiser correcteement les variables affectées d'un point d'interrogation et vérifier la traduction de l'algorithme obtenu en considérant le cas simple où p vaut 1 et q vaut1 et sans propriété de sûreté.*

Question 4.3. *Enoncer la propriété (safemutex) d'exclusion mutuelle de la ressource par rapport aux clients et vérifier cette propriété pour p valant 1 et q valant 1 et aussi pour p=2 et q=2.*

Question 4.4. *Enoncer deux propriétés à vérifier pour montrer*

- *que la ressource R peut être utilisée par un des clients de type 1 (aC1)*
- *que la ressource R peut être utilisée par un des clients de type 2 (aC2)*

Question 4.5. *Enoncer deux propriétés à vérifier pour montrer*

- *que le coordonnateur C peut servir le serveur S1 (grantS1).*
- *que le coordonnateur C peut servir le serveur S2 (grantS2).*

Question 4.6. *(kangourou)*

Quand vous avez testé vos propriéiés, vous avez constaté une alternance très régulère des requtêtes et des services liés à la ressource R. En fait, les processus Client1 et Client2 sont très symétriques et les deux points de contrôle outC1 et outC2 ne permettent pas de casser la symétrie. Une solution est de ralentir le processus client self en plaçant une instruction qui retarde un peu la suite de ce processus. L'instruction skip *est remplacée par une instruction quu ajoute des pas de calcul et donc qui retarde un peu. Comme il faut retarder de manière dissymétrique, une idée est de placer une instruction d'itération qui décroit une variable v d'une unité à chaque itération en l'initialisant par une valeur supposée quelconque et aléatoire. Pour créer l'aléatoire, on peut utiliser un tableau de valeurs pour chaque collection de clients puis d'utiliser ce tableau au début de l'itération.client self.*

```
C1 == [x \in 4..(3+p) |-> m*x % n]
C2 == [x \in (p+4)..(p+3+q) |-> n*x % m]
```

m et n sont de nouvelles constantes à initialiser aussi au moment de la vérification. Ainsi si l'instruction $v := C1[self]$ *est utilisée, elle initialise v d'une valeur pseudo-aléatoire et v est une variable locale à un client self.*

Créer un nouveau fichier aspdil268qbis.tla et recopier le module aspdil268q.tla. Puis modifier les instructions en outC1 et ouC2 pour qu'elles retardent les procesus clients. Valider votre transformation avec p=2 et q=2, en veillant à ce que la trace des messages montre la dissymétrie obtenue.

u

Fin de l'énoncé

Listing 5: pluscalil268q.tla

```
1  ---------- MODULE pluscalil268q ----------
2  EXTENDS Integers, Sequences,TLC
3  ------------------------------------------
4  CONSTANTS p, q
5  ------------------------------------------
6  ASSUME
7  ------------------------------------------
8  (*--algorithm Mutex
9
10 variables
11     R = ?, \* 0 libre, 1 occupÃ©e
12     \* Ã©tats des clients
13     state1 = [i \in 4..(3+p) |-> "idle"],
14     state2 = [i \in (p+4)..(p+3+q) |-> "idle"],
15     \* files FIFO des clients
16     queue1 = <<>>,
17     queue2 = <<>>,
18     \* file FIFO des serveurs
19     coordQ = <<>>,
20     req1 = ?,
21     req2 = ?,
22     grant1 = ?,
23     grant2 = ?;
24
25 \* ================== CLIENTS TYPE 1 ==================
26 process Client1 \in 4..(3+p)
27 begin
28 debut:skip;
29 C1loop: while TRUE do
30         \* idle -> requÃªte
31         l1:state1[self] := "requete";
32         l2:queue1 := Append(queue1, self);
33         \* attendre d'Ãªtre servi
34 ........l3:await state1[self] = "serve";
35 ........\* utilisation ressource
36 ........csin:state1[self] := "critique";
37 ........print <<"Client1", self, "utilise R">>;
38 ........\* libÃ©ration
39 ........csout:state1[self] := "idle"; R:=0;
40 ........outC1:skip;
41 ....end while;
42 end process;
43
44 \* ================== CLIENTS TYPE 2 ==================
45 process Client2 \in (p+4)..(p+3+q)
46 begin
47 debut:skip;
48 C2loop:while TRUE do
49 ........l1:state2[self] := "requete";
50 ........l2:queue2 := Append(queue2, self);
51 ........l3:await state2[self] = "serve";
52 ........csin:state2[self] := "critique";
53 ........print <<"Client2", self, "utilise R">>;
54 ........csout:state2[self] := "idle";  R:=0;
55 ................outC2:skip;
56 ....end while;
57 end process;
58
59
60 ==========================
```

Figure 5: Algorithme du listing 5

Listing 6: pluscalil268qbis.tla

```
1  ---------- MODULE pluscalil268q ----------
2
3  \* suite du module
4  \* ========== SERVEUR S1 ==========
5  process S1 = 1
6  variable c;
7  begin
8  debut:skip;
9  S1loop:while TRUE do
10        l1:await Len(queue1) > 0;
11         l2:req1 := TRUE;
12         l3:coordQ := Append(coordQ, 1);
13         l4:await grant1 = TRUE ;
14         l5:c := Head(queue1);
15         l6:queue1 := Tail(queue1);
16         \* section critique
17         R := 1;
18         state1[c] := "serve";
19         \* print <<"S1 sert Client1", c>>;
20         s1:await R = 0;
21         grant1 := FALSE;
22         req1 := FALSE;
23     end while;
24 end process;
25
26 \* ========== SERVEUR S2 ==========
27 process S2 = 2
28 variable c;
29 begin
30 debut:skip;
31 S2loop:while TRUE do
32         l1:await Len(queue2) > 0;
33         l2:req2 := TRUE;
34         l3:coordQ := Append(coordQ, 2);
35         l4:await grant2 = TRUE;
36         l5:c := Head(queue2);
37         l6:queue2 := Tail(queue2);
38         R := 1;
39         state2[c] := "serve";
40         \* print <<"S2 sert Client2", c>>;
41         s2: await R = 0;
42         grant2 := FALSE;
43         req2 := FALSE;
44     end while;
45 end process;
46
47 \* ========== COORDONNATEUR ==========
48 process C = 3
49 variable s;
50 begin
51 debut:skip;
52 Cloop:while TRUE do
53        l1: await Len(coordQ) > 0 /\ R = 0;
54         l2:s := Head(coordQ);
55         l3:coordQ := Tail(coordQ);
56         l4:if s = 1 then
57              grant1 := TRUE;
58              l5:await grant1 = FALSE;
59         elsif s = 2 then
60              grant2 := TRUE;
61              l6:await grant2 = FALSE;
62         end if;
63         l7:skip;
64     end while;
65 end process;
66
67 end algorithm; *)
68
69
70
71 ==============================
```

Figure 6: Suite de l'algorithme 5 (suite)