Systèmes de transitions - Modélisation TLA

Durée 1h45 - Documents autorisés

12 avril 2011

1 Exercice (6 points)

Soit le module TLA fourni `Test.tla` définissant le système de transitions `Spec`. Les propriétés suivantes, exprimées en LTL ou CTL, sont-elles vérifiées (donner une justification informelle) ?

1. $\Diamond s_1$
2. $\Diamond s_2 \Rightarrow \Diamond s_1$
3. $\Box(s_1 \Rightarrow \Diamond s_2)$
4. $\exists\Box s_0$
5. $\exists\Diamond\exists\Box s_1$
6. $\forall\Diamond s_2 \Rightarrow \forall\Diamond s_1$

Les exécutions sont : ${s_0}^\omega$ et ${s_0}^* \to s_1 \to {s_2}^\omega$ (du fait de WF(Trans1), ${s_0}^* \to {s_1}^\omega$ n'est pas une exécution possible).

1. *non (${s_0}^\omega$)*
2. *oui ($= \Box\neg s_2 \vee \Diamond s_1$)*
3. *oui (équité faible)*
4. *oui*
5. *non (s_1 nécessairement transitioire)*
6. *oui ($= \exists\Box\neg s_2 \vee \forall\Diamond s_1$)*

1.1 Module fourni : `Test.tla`

```
---- MODULE Test ----
VARIABLE etat

Trans0 == etat = "s0" /\ etat' = "s1"

Trans1 == etat = "s1" /\ etat' = "s2"

Spec == /\ etat = "s0" /\ [] [ Trans0 \/ Trans1 ]_<<etat>>
        /\ WF_<<etat>>(Trans1)
================================================================
```

2 Problème : algorithme auto-stabilisant (14 points)

Un algorithme auto-stabilisant est un algorithme qui, quel que soit l'état initial et les transitions choisies, finit nécessairement par atteindre un ensemble d'états favorables (i.e. vérifiant une propriété intéressante) et ne plus quitter cet ensemble.

On considère un système constitué de N processus. Chaque processus a une instruction (instruction applicative sans importance pour nous) qu'il exécute quand il possède un *privilège*. Les états favorables sont ceux où un seul processus à la fois possède un privilège.

Dans l'algorithme de Dijkstra, les N processus sont disposés en anneau et chacun a une variable d'état entière prenant valeur dans $0..V-1$. L'état initial est quelconque, tous les états possibles étant initiaux. Initialement, il peut donc y avoir plusieurs processus privilégiés. L'objectif de l'exercice est de montrer que l'algorithme suivant est auto-stabilisant en convergeant nécessairement vers un ensemble d'états où il n'y a toujours qu'un seul processus privilégié.

Le pseudo-code pour le processus P_i est une répétition infinie de :

```
if (i=0) then if  (val[i]  = val[N-1])    // privilège de P0
              then // instruction privilégiée de P0
                   val[i] := (val[i]+1)%V
              endif
         else if  (val[i] /= val[i-1])    // privilège de Pi, i/=0
              then // instruction privilégiée de Pi, i/=0
                   val[i] := val[i-1]
              endif
endif
```

Un squelette de module TLA `Dijkstra.tla` est fourni en 2.5.

2.1 Spécification

Caractérisez en TLA :

1. Le prédicat d'état `privilege(`i`)` spécifiant la possibilité pour le processus i d'exécuter son instruction.

   $privilege(i) \triangleq (i = 0 \land val[i] = val[N-1]) \lor (i \neq 0 \land val[i] \neq val[i-1])$

2. L'expression `Nbprivileges` définissant le nombre de processus possédant le privilège simultanément dans un état donné.

   $Nbprivileges \triangleq Cardinality(\{i \in Proc : privilege(i)\})$

3. Le prédicat d'état `Favorable` spécifiant les états favorables de l'algorithme.

   $Favorable \triangleq Nbprivileges = 1$

4. La propriété `TransitoirePrivilege` spécifiant que si un processus possédant le privilège exécute son instruction, il perdra son privilège (propriété simple ne nécessitant ni $\Diamond$ ni $\leadsto$).

   $instruction(i) \triangleq (i = 0 \land val[i]' = (val[i]+1)\%V) \lor (i \neq 0 \land val[i]' = val[i-1])$

   $TransitoirePrivilege \triangleq \forall i : \Box(privilege(i) \land instruction(i) \Rightarrow \neg privilege(i)')$

5. La propriété `DecroissancePrivilege` spécifiant que le nombre de privilèges simultanés ne peut que décroître.

   $DecroissancePrivilege \triangleq \forall k : \Box(Nbprivileges = k \Rightarrow \Box(Nbprivileges \leq k))$

   $DecroissancePrivilege \triangleq \Box(Nbprivileges' <= Nbprivileges)$

6. La propriété `Inevitable` spécifiant qu'on finit toujours par atteindre un état où toutes les valeurs des processus sont égales.

   $Inevitable \triangleq \Box\Diamond(\forall i, j : val[i] = val[j])$

7. La propriété `VivacitePrivilege` spécifiant l'absence de famine des processus pour l'accès au privilège.
   $VivacitePrivilege \triangleq \forall i : \Box\Diamond privilege(i)$
8. La propriété `AutoStabilisation` spécifiant la propriété d'auto-stabilisation, comme quoi on finit nécessairement par atteindre un ensemble d'états où un seul processus est privilégié et ne plus quitter cet ensemble.
   $AutoStabilisation \triangleq \Diamond Favorable \wedge \Box(Favorable \Rightarrow \Box Favorable)$
   (moins fort : $AutoStabilisation \triangleq \Diamond\Box Favorable$)

2.2 Module simple

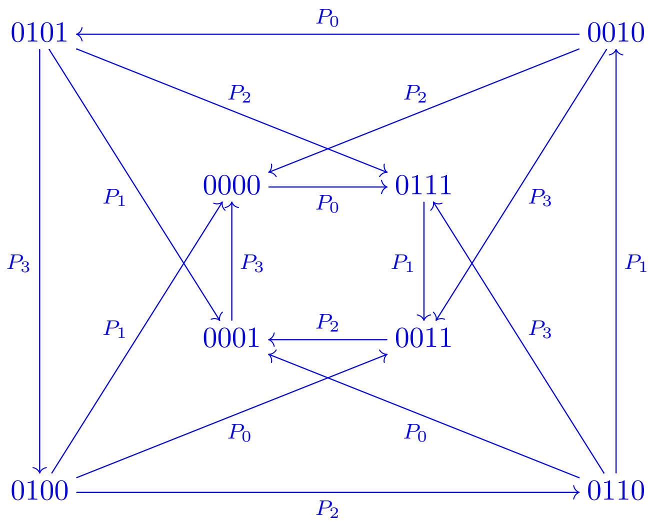
On modélise chaque processus par une seule action TLA, représentant l'exécution **atomique** du pseudo-code.

9. Complétez le prédicat `Init` définissant les états initiaux possibles. Les valeurs initiales proposées par les processus devront être quelconques.
   $Init \triangleq val \in [Proc \rightarrow Valeur]$
10. Complétez l'action `PseudoCode(i)` définissant l'exécution du processus i.
   $$PseudoCode(i) \triangleq privilege(i) \ \wedge (i = 0 \Rightarrow val = [val \ except \ ![i] = (v[i] + 1)\%V]) \\ \wedge (i \neq 0 \Rightarrow val' = [val \ except \ ![i] = v[i-1]])$$
11. Donnez l'équité minimale `Fairness(i)` permettant de simuler complètement l'exécution du processus i.
   $Fairness(i) \triangleq WF_{val}(PseudoCode(i))$
12. Donnez l'expression `Spec` caractérisant les états initiaux, les transitions possibles des processus, ainsi que l'équité.
   $Next \triangleq \exists i \in Proc : PseudoCode(i)$
   $Fairness \triangleq \forall i \in Proc : Fairness(i)$
   $Spec \triangleq Init \wedge \Box[Next]_{val} \wedge Fairness$

2.3 Vérification

On souhaite s'assurer que la contrainte $V \geq N - 1$ est nécessaire. Pour cela, on montre que certaines propriétés souhaitées sont invalides lorque $V < N-1$. On admet qu'incrémenter simultanément toutes les valeurs des processus de k (modulo V) laisse le système globalement inchangé. Ainsi, les états de l'ensemble $\{[i \in Proc \mapsto (val[i] + k)\%V] : k \in Valeur\}$ sont équivalents et doivent être fusionnés, afin de simplifier la construction du graphe des exécutions.

13. Dessinez le graphe des exécutions pour $V = 2, N = 4$ (8 états, 16 transitions). Vous veillerez à indiquer sur les transitions le processus ayant exécuté son instruction. Les états sont $[0,0,0,0], [0,0,0,1], [0,0,1,0], [0,1,0,0], [0,0,1,1], [0,1,1,0], [0,1,0,1], [0,1,1,1]$.



Note : tous les états étant initiaux, cela n'a pas été indiqué sur le graphe. De même les boucles de bégaiement n'ont pas été dessinées.

14. Justifiez en examinant le graphe, la validité (ou non) de la propriété `VivacitePrivilege`.

    L'équité minimale interdit le bégaiement et il n'y a que deux cycles : $[0,0,0,0] \xrightarrow{P_0} [0,1,1,1] \xrightarrow{P_1} [0,0,1,1] \xrightarrow{P_2} [0,0,0,1] \xrightarrow{P_3} [0,0,0,0]$ *et* $[0,0,1,0] \xrightarrow{P_0} [0,1,0,1] \xrightarrow{P_3} [0,1,0,0] \xrightarrow{P_2} [0,1,1,0] \xrightarrow{P_1} [0,1,0,1]$. *Ces deux cycles incluent tous les processus, et toutes les autres transitions conduisent du deuxième cycle au premier. Donc la vivacité est vraie*

15. Faites de même pour la propriété `AutoStabilisation`.

    Vu qu'on peut rester infiniment dans le deuxième cycle, non favorable, la propriété est fausse.

2.4 Preuve

On se replace ici dans le cas général, pour $V \geq N-1$.

16. Prouvez la propriété `TransitoirePrivilege`.

    L'instruction invalide trivialement la garde définissant la possession du privilège.

17. Justifiez, notamment par des considérations sur l'équité, que tout processus possédant le privilège finira par le perdre.

    Il y a équivalence entre « posséder le privilège » et « l'action du processus est faisable ». Quand un processus P possède le privilège, et comme son action est en équité faible :
    - *soit l'action est continûment faisable, donc elle est exécutée, et vu que `TransitoirePrivilege` est vraie, le processus perd le privilège ;*
    - *soit l'action n'est pas continûment faisable et donc le processus finit par perdre le privilège.*

18. Justifiez que, si un processus P_i ne change jamais de valeur, alors tous les $P_j, i < j < N$ finiront pas prendre cette valeur.

    Supposons que P_i possède la valeur v. Si P_{i+1} possède une valeur différente de v, il possède le privilège. Vu le résultat précédent, il doit finir par le perdre et donc prendre la *même valeur que P_i (code de son action). Une fois que P_{i+1} a la même valeur que P_i, il ne peut plus en changer tant que P_i n'en change pas. Par récurrence, on obtient que pour $N > j > i$, P_j finira par prendre la valeur de P_i et n'en changera pas.*

19. En raisonnant par l'absurde, prouvez que le processus 0 finit par acquérir nécessairement le privilège depuis tout état initial, et donc infiniment souvent.

    Supposons que P_0 prenne une valeur v et n'en change plus. D'après le résultat précédent P_{N-1} finira par prendre cette valeur. Les valeurs de P_0 et P_{N-1} seront égales, P_0 aura alors le privilège et son action sera faisable. Vu le résultat pénultième, il perdra le privilège en prenant la valeur $v+1 \bmod V$. Ceci est vrai $\forall v \in 0..V-1$, donc pour tout état initial de P_0. Comme les résultats précédents sont vrais pour des états quelconques des P_i, la propriété est vraie depuis un état quelconque.

2.5 Squelette fourni : `Dijkstra.tla`

```
---------------- MODULE dijkstra ----------------
EXTENDS Naturals

CONSTANT  N,   \* nombre de processus
          V    \* nombre de valeurs
VARIABLES val

Valeur == 0..V-1
Proc == 0..N-1

gauche(i) == (i+N-1) % N
succ(v) == (v+1) % V

TypeInvariant == val \in [ Proc -> Valeur ]

\* PROPRIÉTÉS ...
----------------------------------------------------------------
Init == \* À COMPLETER
PseudoCode(i) == \* À COMPLETER
Fairness(i) == \* À COMPLETER
Spec == \* À COMPLETER
================================================================
```