Systèmes de transitions - Modélisation TLA$^+$

Durée 1h30 - Documents autorisés

14 avril 2023

1 Questions de cours (4 points)

Soit quatre variables w, x, y, z. w est un entier, x est une fonction dans $[Nat \rightarrow Nat]$, et y et z sont des ensembles d'entiers.

1. Donner une action qui change w en une valeur quelconque de y.
   $w' \in y$ *ou* $\exists v \in y : w' = v$
   $(\wedge$ *UNCHANGED* $\langle x, y, z \rangle)$
2. Donner une propriété temporelle qui dit que le domaine de x est toujours inclus dans y.
   $\Box((\textit{DOMAIN } x) \subseteq y)$
3. Donner un prédicat qui dit que w est dans y et dans z.
   $w \in y \wedge w \in z$ *ou* $w \in y \cap z$
4. Donner une action qui change la valeur de x en 0 pour prendre la valeur de w.
   $x' = [x \textit{ EXCEPT } ![0] = w]$ $(\wedge$ *UNCHANGED* $\langle w, y, z \rangle)$

2 Exercice (4 points)

Soit le système S :

$$\begin{tikzcd} {} \arrow[r] & s_0 \arrow[r, bend left] & s_1 \arrow[l, bend left] \arrow[r] & s_2 \arrow[r, bend left] & s_3 \arrow[l, bend left] \arrow[ll, bend left] \arrow[loop above] \end{tikzcd}$$

Indiquer si les propriétés suivantes, exprimées en logique LTL ou CTL, sont vérifiées. Justifier les réponses (argumentaire ou contre-exemple).

	sans équité	$WF(s_3, s_1)$	$SF(s_3, s_1) \wedge SF(s_1, s_0)$
$\Diamond s_2$			
$\Box\Diamond s_1$			
$s_2 \leadsto s_1$			
$\exists\Diamond\exists\Box s_3$			
$\forall\Box\exists\Diamond s_0$			
$\Box(s_1 \Rightarrow (s_1 \mathcal{U} s_0))$			

Notation : 1/4 point par bonne réponse justifiée ; -1/4 par mauvaise réponse oui absence d'explication ou explication erronée.

	sans équité	$WF(s_3, s_1)$	$SF(s_3, s_1) \wedge SF(s_1, s_0)$
$\Diamond s_2$	*non* $(s_0 \rightarrow s_1)^\omega$	*non (idem)*	*non (idem)*
$\Box\Diamond s_1$	*non* $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow {s_3}^\omega$	*non* $s_0 \rightarrow s_1 \rightarrow (s_2 \rightarrow s_3)^\omega$	*oui (1)*
$s_2 \leadsto s_1$	*non* $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow {s_3}^\omega$	*non* $s_0 \rightarrow s_1 \rightarrow (s_2 \rightarrow s_3)^\omega$	*oui (1)*
$\exists\Diamond\exists\Box s_3$	*oui (2)*	*non (1)*	*non (1)*
$\forall\Box\exists\Diamond s_0$	*oui (3)*	$\diagdown$	*oui (3)*
$\Box(s_1 \Rightarrow (s_1 \mathcal{U} s_0))$	*non*	$\diagdown$	*non (4)*

1. *L'équité faible supprime les exécutions* $\rightarrow {s_3}^\omega$*. Les équités fortes suppriment, entre autres, les exécutions* $\rightarrow {s_3}^\omega$ *et* $\rightarrow (s_2 \rightarrow s_3)^\omega$ *: toute exécution passe nécessairement infiniment souvent par* s_1*.*
2. *= accessibilité de* ${s_3}^\omega$ *depuis l'état initial.*
3. *= accessibilité de* s_0 *depuis n'importe quel état.*
4. *En* s_1*, on peut toujours aller en* s_2 *: l'exécution* $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ *est valide même avec l'équité. L'équité obliger seulement à aller en* s_0 *infiniment souvent (*$\Box\Diamond(s_1 \Rightarrow (s_1 \mathcal{U} s_0))$ *est vrai avec l'équité).*

3 Problème de l'ensemble indépendant maximal (12 points[1])

Dans un graphe non orienté, un ensemble *stable* ou *indépendant* (en anglais *independent set*) est un sous-ensemble de nœuds du graphe dont aucun n'est adjacent à un autre. Formellement, pour un graphe (V, E) où V sont les nœuds (*vertices*) et E les arêtes (*Edges*), un sous-ensemble S de V est un indépendant si pour tout couple de sommets de S, il n'existe pas d'arête entre eux : $\forall u, v \in S : (u, v) \notin E$. Un ensemble indépendant est *maximal* (*maximal independent set* ou *MIS*) s'il ne peut pas être étendu en ajoutant d'autres nœuds.

Par exemple, pour le graphe $\begin{array}{ccccc} & 1 & & & \\ 2 \diagup\!\!\diagdown & & \diagdown\!\!\diagup\, 4 & — & 5 \\ & 3 & & & \end{array}$, nous avons les ensembles indépendants maximaux suivants : $\{1, 3, 5\}$, $\{2, 4\}$, $\{2, 5\}$. Les ensembles $\{1\}$ ou $\{1, 3\}$ ou $\{1, 5\}$ sont indépendants mais pas maximaux. Noter qu'un indépendant maximal n'est pas nécessairement optimal (par exemple $\{2, 4\}$).

3.1 Algorithme séquentiel

On considère l'algorithme suivant. Il consiste à prendre un nœud au hasard dans le graphe, à l'ajouter à l'ensemble en construction et à éliminer ce nœud et tous ses voisins. Ceci est répété tant qu'il reste des nœuds.

$$
\begin{array}{l}
\mathit{Nodes} = V \\
\mathit{MIS} = \emptyset \\
\text{repeat} \\
\quad \text{pick } v \in \mathit{Nodes} \\
\quad \mathit{MIS} := \mathit{MIS} \cup \{v\} \\
\quad \mathit{Nodes} := (\mathit{Nodes} \setminus \{v\}) \setminus \mathit{neighbors}(v) \\
\text{until } \mathit{Nodes} = \emptyset
\end{array}
$$

1. Toutes les questions valent autant.

Une modélisation TLA$^+$ de cet algorithme est fourni en fin de sujet (annexe A). On fixe le nombre de nœuds (N). Une arête (élément de E) est représentée par un ensemble à deux éléments, les deux nœuds que l'arête relie. Chaque exécution débute avec un graphe quelconque (appartenant à *AllGraphs* qui est l'ensemble de tous les graphes possibles avec N nœuds). Noter que E (l'ensemble des arêtes) est une constante tout du long d'une exécution.

3.1.1 Spécification

Exprimer en LTL ou CTL les propriétés suivantes (qui ne sont pas nécessairement vérifiées par le modèle TLA$^+$) :

1. `MISPasVoisins` : MIS ne peut jamais contenir deux nœuds qui sont voisins.
   $MISPasVoisins \triangleq \Box(Independent(MIS))$
   $MISPasVoisins \triangleq \Box(\forall u, v \in MIS : (u, v) \notin E)$
   $MISPasVoisins \triangleq \Box(\forall u, v \in MIS : u \notin neighbors(v))$
   (c'est ***`MISIndependent`*** *de l'examen 21)*
2. `Terminaison` : l'algorithme termine, c'est-à-dire que l'on finit par atteindre un état où *Nodes* est vide.
   $Terminaison \triangleq \Diamond(Nodes = \emptyset)$
3. `ToutFini` : on atteint un état permanent où tous les nœuds ont été traités et l'ensemble *MIS* obtenu est indépendant maximal.
   $ToutFini \triangleq \Diamond\Box(Nodes = \emptyset \land Independent(MIS) \land Maximal(MIS))$

3.1.2 Équité

4. Avec la spécification TLA$^+$ fournie, indiquer si les propriétés `MISPasVoisins` et `Terminaison` sont vérifiées, en justifiant votre réponse.
   `MISPasVoisins` *: oui : c'est un invariant ; il est vrai dans l'état initial et l'action Next en maintient la véracité vu que Nodes ne contient que les nœuds non voisins des nœuds déjà dans MIS.*
   `Terminaison` *: non, pas d'équité, on peut bégayer dans l'état initial.*
5. Énoncer l'équité minimale nécessaire pour que ces deux propriétés soient vérifiées.
   $WF_{vars}(Next)$ *ou* $\forall v \in V : WF_{vars}(Action(v))$

$\longrightarrow$

3.2 Algorithme distribué

Dans la version distribué, chaque nœud échange des messages avec ses voisins. Un message est un triplet $\langle type, from, to \rangle$ où *from* et *to* sont les nœuds respectivement émetteur et destinataire, et *type* peut prendre les valeurs "*join*", "*ack*" ou "*token*". Pour éviter que deux voisins ne tentent simultanément d'intégrer l'ensemble indépendant, un seul nœud n'est actif à un moment donné et il passe le jeton au suivant quand il a terminé. Quand un nœud v reçoit le jeton et qu'il a fini, il le transmet au suivant. Quand v reçoit le jeton et qu'il est actif ($\neg Done[v]$), il intègre l'ensemble *MIS* et il envoie "*join*" à tous ses voisins, puis il attend de recevoir un "*ack*" de chacun de ses voisins ; à ce moment-là, il marque qu'il a fini et passe le jeton au suivant. Quand un site reçoit un message "*join*", il note qu'il a fini (sans être dans l'ensemble *MIS*) et répond "*ack*". Un squelette incomplet est fourni dans l'annexe B.

3.2.1 Modélisation

6. Compléter le prédicat de transition *Next*.
   $$\begin{array}{l} Next \triangleq \exists v \in V : \\ \qquad \lor\ SendToken(v) \lor ReceiveTokenNotDone(v) \lor ReceiveTokenDone(v) \\ \qquad \lor\ \exists w \in V : ReceiveJoin(v, w) \lor ReceiveAck(v, w) \end{array}$$
7. Compléter l'action *ReceiveJoin*.
   $$\begin{array}{l} ReceiveJoin(from, to) \triangleq \\ \quad \land \langle \text{"}join\text{"}, from, to \rangle \in network \\ \quad \land network' = (network \setminus \{\langle \text{"}join\text{"}, from, to \rangle\}) \cup \{\langle \text{"}ack\text{"}, to, from \rangle\} \\ \quad \land Done' = [Done \ \text{EXCEPT} \ ![to] = \text{TRUE}] \\ \quad \land \text{UNCHANGED} \ \langle Edge, Ack, MIS \rangle \end{array}$$

3.2.2 Spécification

Exprimer en LTL ou CTL les propriétés suivantes :

8. `AuPlusUnJeton` : il y a au plus un message "*token*" en transit.
   $AuPlusUnJeton \triangleq \Box(Cardinality(\{m \in network : m[0] = \text{"}token\text{"}\}) \leq 1)$
9. `DoneStable` : *Done* est stable : une fois vrai, il reste vrai.
   $DoneStable \triangleq \forall v \in V : \Box(Done[v] \Rightarrow \Box(Done[v]))$
10. `JetonPasFini` : pour un site quelconque, il est possible qu'il reçoive le jeton alors qu'il est actif (il n'a pas encore fini).
    $JetonPasFini \triangleq \forall v \in V : \exists\Diamond(\neg Done[v] \land \exists m \in network : m[0] = \text{"}token\text{"} \land m[2] = v)$
    $JetonPasFini \triangleq \forall v \in V : \exists\Diamond(\neg Done[v] \land \exists w \in V : \langle \text{"}token\text{"}, w, v \rangle \in network)$

3.3 Vérification

11. Expliquer informellement comment vérifier la propriété `MISPasVoisins` (indifféremment sur `maxindepsetseq` ou `maxindepsetdist`).
    C'est un invariant : construire l'ensemble des états accessibles et vérifier le prédicat dans chaque état.
12. Expliquer informellement comment vérifier la propriété `JetonPasFini`.
    C'est une propriété d'accessibilité : construire l'ensemble des états accessibles et vérifier qu'il existe, pour chaque nœud v, au moins un état qui vérifie le prédicat. Ou appliquer l'algorithme de marquage du cours pour une propriété CTL $\exists\Diamond$.

A Version séquentielle : `maxindepsetseq.tla`

```
---------------------------- MODULE maxindepsetseq ----------------------------
EXTENDS Naturals, FiniteSets
CONSTANT N   \* size of the graphs
V ≜ 1 .. N   \* node identifiers

\* The set of all possible graphs with V nodes.
\* An edge is a set of two nodes (a node cannot have an edge to itself)
\* A graph is a set of edges, i.e. a set of sets of nodes.
AllGraphs ≜ SUBSET {e ∈ SUBSET V : Cardinality(e) = 2}

VARIABLES E, Nodes, MIS
vars ≜ ⟨E, Nodes, MIS⟩

neighbors(v) ≜ (UNION {e ∈ E : v ∈ e}) \ {v}

Init ≜
  ∧ E ∈ AllGraphs   \* E (the graph) is a constant in an execution
  ∧ Nodes = V
  ∧ MIS = {}

Action(v) ≜
    ∧ MIS' = MIS ∪ {v}
    ∧ Nodes' = (Nodes \ {v}) \ neighbors(v)
    ∧ UNCHANGED E

Next ≜ ∃ v ∈ Nodes : Action(v)
Spec ≜ Init ∧ □[Next]_vars
--------------------------------------------------------------------------------
TypeOk ≜ ∧ E ∈ SUBSET SUBSET V
         ∧ Nodes ∈ SUBSET V
         ∧ MIS ∈ SUBSET V

Independent(s) ≜   \* no two neighbors in the set
    ∀ v ∈ s : neighbors(v) ∩ s = {}

Maximal(s) ≜   \* the set cannot be expanded without losing its independence.
    ∀ v ∈ V : v ∈ s ∨ ¬Independent(s ∪ {v})
================================================================================
```

B Version distribuée : `maxindepsetdist.tla`

MODULE *maxindepsetdist*

$\text{EXTENDS } Naturals,\ FiniteSets$
$\text{CONSTANT } N$
$V \triangleq 1 \,..\, N$
$AnyGraph \triangleq \text{SUBSET } \{e \in \text{SUBSET } V : Cardinality(e) = 2\}$

$\text{VARIABLES } Edge,\ Done,\ Ack,\ MIS,\ network$
$vars \triangleq \langle Edge,\ Done,\ Ack,\ MIS,\ network\rangle$
$neighbors(v) \triangleq (\text{UNION } \{e \in Edge : v \in e\}) \setminus \{v\}$

$nextturn(v) \triangleq (v \% N) + 1$ any order could work

$Init \triangleq$
$\quad \land\ Edge \in AnyGraph$
$\quad \land\ Done = [v \in V \mapsto \text{FALSE}]$
$\quad \land\ Ack = [v \in V \mapsto \{\}]$
$\quad \land\ MIS = \{\}$
$\quad \land\ \exists\, v \in V : network = \{\langle \text{“token”},\ v,\ nextturn(v)\rangle\}$ initially, an arbitrary node will receive the token

$ReceiveTokenDone(v) \triangleq$
$\quad \exists\, w \in V : \langle \text{“token”},\ w,\ v\rangle \in network$
$\qquad \land\ Done[v]$
$\qquad \land\ network' = (network \setminus \{\langle \text{“token”},\ w,\ v\rangle\}) \cup \{\langle \text{“token”},\ v,\ nextturn(v)\rangle\}$
$\qquad \land\ \text{UNCHANGED } \langle Edge,\ Done,\ MIS,\ Ack\rangle$

$ReceiveTokenNotDone(v) \triangleq$
$\quad \exists\, w \in V : \langle \text{“token”},\ w,\ v\rangle \in network$
$\qquad \land\ \neg Done[v]$
$\qquad \land\ MIS' = MIS \cup \{v\}$
$\qquad \land\ network' = (network \setminus \{\langle \text{“token”},\ w,\ v\rangle\}) \cup \{\langle \text{“join”},\ v,\ n\rangle : n \in neighbors(v)\}$
$\qquad \land\ \text{UNCHANGED } \langle Edge,\ Ack,\ Done\rangle$

$ReceiveJoin(from,\ to) \triangleq TODO$

$ReceiveAck(from,\ to) \triangleq$
$\quad \land\ \langle \text{“ack”},\ from,\ to\rangle \in network$
$\quad \land\ Ack' = [Ack \text{ EXCEPT } ![to] = Ack[to] \cup \{from\}]$
$\quad \land\ network' = network \setminus \{\langle \text{“ack”},\ from,\ to\rangle\}$
$\quad \land\ \text{UNCHANGED } \langle Edge,\ Done,\ MIS\rangle$

$SendToken(v) \triangleq$
$\quad \land\ v \in MIS$
$\quad \land\ \neg Done[v]$
$\quad \land\ Ack[v] = neighbors(v)$
$\quad \land\ network' = network \cup \{\langle \text{“token”},\ v,\ nextturn(v)\rangle\}$
$\quad \land\ Done' = [Done \text{ EXCEPT } ![v] = \text{TRUE}]$
$\quad \land\ \text{UNCHANGED } \langle Edge,\ Ack,\ MIS\rangle$

$Next \triangleq TODO$

$Spec \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(Next)$