Systèmes de transitions - Modélisation TLA$^+$

Durée 1h30 - Documents autorisés

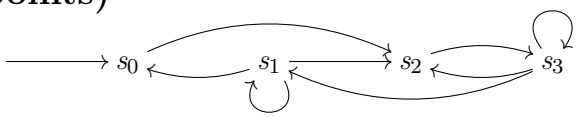
2 avril 2024

1 Questions de cours (4 points)

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

1. Donner une action qui change w en l'image par x d'une des valeurs de son domaine.
   $\exists v \in (\textit{DOMAIN } x) : w' = x[v]$ $(\wedge \textit{ UNCHANGED } \langle x, y, z\rangle)$
   $w' \in \{x[v] : v \in (\textit{DOMAIN } x)\}$ $(\wedge \textit{ UNCHANGED } \langle x, y, z\rangle)$
2. Donner une propriété temporelle qui dit que le co-domaine de x est toujours inclus dans y.
   $\Box(\forall e \in \textit{DOMAIN } x : x[e] \in y)$
   $\Box(\{x[e] : e \in \textit{DOMAIN } x\} \subseteq y)$
3. Donner un prédicat qui dit que w est dans y ou (exclusif) dans z.
   $(w \in y \wedge w \notin z) \vee (w \notin y \wedge w \in z)$
4. Donner une action qui permute les valeurs de x en 0 et 1.
   $x' = [x \textit{ EXCEPT } ![0] = x[1], ![1] = x[0]]$ $(\wedge \textit{ UNCHANGED } \langle w, y, z\rangle)$

2 Exercice (4 points)

Soit le système S : 

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

	sans équité	$WF(s_1, s_0) \wedge WF(s_3, s_1)$	$SF(s_1, s_0) \wedge SF(s_3, s_1)$
$\bigcirc\Diamond s_0$			
$\Box\Diamond s_0$			
$s_3 \rightsquigarrow s_0$			
$\exists\Diamond\exists\Box s_1$			
$\exists\bigcirc\exists\Diamond s_0$		\	
$\forall\Box(s_3 \Rightarrow \exists\Diamond s_0)$		\	

Notation : 1/4 point par bonne réponse justifiée ; 0 point par mauvaise réponse ou absence d'explication ou explication erronée.

	sans équité	$WF(s_1, s_0) \wedge WF(s_3, s_1)$	$SF(s_1, s_0) \wedge SF(s_3, s_1)$
$\bigcirc\Diamond s_0$	*non* $s_0 \to s_2 \to {s_3}^\omega$	*non* $s_0 \to (s_2 \to s_3)^\omega$	*oui (2)*
$\Box\Diamond s_0$	*non* $s_0 \to s_2 \to {s_3}^\omega$	*non* $s_0 \to (s_2 \to s_3)^\omega$	*oui (2)*
$s_3 \rightsquigarrow s_0$	*non* $s_0 \to s_2 \to {s_3}^\omega$	*non* $s_0 \to (s_2 \to s_3)^\omega$	*oui (2)*
$\exists\Diamond\exists\Box s_1$	*oui (3)*	*non (1)*	*non (1) ou (2)*
$\exists\bigcirc\exists\Diamond s_0$	*oui (4)*	$\diagdown$	*oui (4)*
$\forall\Box(s_3 \Rightarrow \exists\Diamond s_0)$	*oui (5)*	$\diagdown$	*oui (5)*

1. *L'équité faible supprime les exécutions* $\to {s_1}^\omega$ *(mais ne force pas la transition* $s_1 \to s_0$ *:* $s_0 \to (s_1 \to s_2 \to s_3)^\omega$ *est légale) et* $\to {s_3}^\omega$ *(mais ne force pas la transition* $s_3 \to s_1$ *:* $\to (s_3 \to s_2)^\omega$ *est légale).*
2. *Les équités fortes forcent à passer infiniment souvent en* s_1 *et* s_0 *(et donc en* s_2*, seul état accessible depuis* s_0*) : elles suppriment entre autres les exécutions* $\to {s_3}^\omega$*,* $\to (s_2 \to s_3)^\omega$*,* $\to (s_3 \to s_1 \to s_2)^\omega$ *(et avec le bégaiement).*
3. $=$ *accessibilité de* ${s_1}^\omega$ *depuis l'état initial.*
4. $=$ *possibilité de revenir une fois en* s_0*.*
5. $=$ *accessibilité de* s_0 *depuis* s_3*.*

3 Problème : gestion d'un carrfeour (12 points [1])

On s'intéresse à un algorithme de gestion de carrefour. Un carrefour est constitué de plusieurs voies dont certaines se croisent. Un véhicule se présente à l'entrée du carrefour sur une voie, il attend l'autorisation, il traverse ensuite et sort du carrefour. Pour obtenir l'autorisation, quand un véhicule se présente, il envoie un message de requête à chacun des véhicules déjà présents (en train de traverser ou en attente d'accès). Un véhicule qui reçoit un message de requête répond ok s'il n'est pas à l'intérieur du carrefour ou si sa voie n'est pas en conflit avec celle du véhicule demandeur. Quand un véhicule a reçu l'autorisation de tous les véhicules présents à son arrivée, il peut traverser.

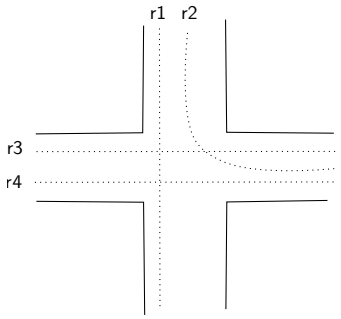


FIGURE 1 – Exemple de carrefour

Pour assurer l'équité, les messages sont traités séquentiellement dans l'ordre de production. Ainsi un véhicule B arrivé après un véhicule A verra ses messages de requêtes traités après tous les messages de A.

Une modélisation TLA$^+$ de cet algorithme est fourni en fin de sujet (annexe A). Un message est une structure avec quatre champs : *kind* qui indique si c'est un message de requête ou d'autorisation ; *from* et *to* indiquent le véhicule émetteur et le véhicule destinataire ; *route* indique la route concernée par le message. Le réseau est modélisé par la variable *network* qui contient une séquence de messages.

Les transitions sont les suivantes :

Request(v, r) : un véhicule v se présente pour emprunter la voie r. Il envoie un message à chacun des véhicules déjà présents et note dans *expectedAck* qu'il attend l'acquittement de ceux-ci.

AllowRequest(v) : un véhicule v prend le message en tête, à condition que ce soit une requête pour lui et qu'il ne soit pas en conflit avec la demande, et renvoie un acquittement au demandeur (l'émetteur).

ReceiveAck(v) : un véhicule v prend le message en tête, à condition que ce soit un acquittement pour lui, et enlève l'émetteur de *expectedAck*.

Enter(v) : un véhicule demandeur v passe dans l'état *Crossing* à condition qu'il ait reçu tous les acquittements nécessaires.

Leave(v) : un véhicule v finit sa traversée et quitte le carrefour.

Pour illustrer, on peut imaginer un système avec trois véhicules, quatre routes et un graphe de conflits où r_1 conflicte avec r_3 et r_4, et r_2 conflicte avec r_3 (la relation de conflit est symétrique), comme sur la figure 1 :

```
Vehicle ≜ {"v1", "v2", "v3"}
Route ≜ {"r1", "r2", "r3", "r4"}
Conflict ≜   "r1" :> {"r3", "r4"}
          @@ "r2" :> {"r3"}
          @@ "r3" :> {"r1", "r2"}
          @@ "r4" :> {"r1"}
```

1. Toutes les questions valent autant.

3.0.1 Spécification

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

1. `NoAccident` : il n'y a jamais deux véhicules simultanément en train de traverser et sur des routes conflictuelles.
   $NoAccident \triangleq \forall v1, v2 \in Vehicle : \Box(\neg(state[v1] = Crossing \wedge state[v2] = Crossing \wedge route[v1] \in Conflict[route[v2]]))$
2. `AbsenceDeBouchon` : tout véhicule demandeur finit par obtenir l'autorisation de traverser
   $AbsenceDeBouchon \triangleq \forall v \in Vehicle : state[v] = Requesting \rightsquigarrow state[v] = Crossing$
   $AbsenceDeBouchon \triangleq \forall v \in Vehicle : state[v] = Requesting \rightsquigarrow state[v] = Idle$ *(bof)*
   $AbsenceDeBouchon \triangleq \forall v \in Vehicle : state[v] = Requesting \rightsquigarrow expectedAck = \emptyset$
3. `EmptyNetwork` : le réseau est infiniment souvent vide / il y a infiniment souvent aucun message en transit.
   $EmptyNetwork \triangleq \Box\Diamond(Len(network) = 0)$
   $EmptyNetwork \triangleq \Box\Diamond(network = \langle\rangle)$
4. `Multicrossing` : il est possible que plusieurs véhicules traversent simultanément le carrefour.
   $Multicrossing \triangleq \exists\Diamond(Cardinality(\{v \in Vehicle : state[v] = Crossing\}) \geq 2)$
5. `NoLoss` : un message de requête reste en transit tant que le message d'acquittement correspondant n'est pas envoyé.
   $NoLoss \triangleq \Box(\forall m \in network : m.kind = \text{“}Req\text{”} \Rightarrow (m.kind = \text{“}Req\text{”} \ \mathcal{W} \ (\exists m2 \in network : m2.kind = \text{“}Ack\text{”} \wedge m2.to = m.from \wedge m2.route = m.route)))$
   $NoLoss \triangleq \Box(\forall m \in network : m.kind = \text{“}Req\text{”} \Rightarrow (m.kind = \text{“}Req\text{”} \ \mathcal{U} \ (\exists m2 \in network : m2.kind = \text{“}Ack\text{”} \wedge m2.to = m.from \wedge m2.route = m.route)))$
   (la formulation est ambiguë)
   (je triche en utilisant $\in$ avec une séquence)

3.0.2 Modélisation

6. Compléter le prédicat de transition *Next*.
   $Next \triangleq \exists\, v \in Vehicle :$
   $\quad \vee \exists\, r \in Route : Request(v, r)$
   $\quad \vee Enter(v)$
   $\quad \vee Leave(v)$
   $\quad \vee AllowRequest(v)$
   $\quad \vee ReceiveAck(v)$
7. Compléter l'action *Enter*.
   $Enter(v) \triangleq$
   $\quad \wedge state[v] = Requesting$
   $\quad \wedge expectedAck[v] = \{\}$
   $\quad \wedge state' = [state \ \text{EXCEPT} \ ![v] = Crossing]$
   $\quad \wedge \text{UNCHANGED} \ \langle route, network, expectedAck\rangle$

8. Compléter l'action *ReceiveAck*.
   $ReceiveAck(v) \triangleq$
   $\land\ Len(network) > 0$
   $\land$ LET $m \triangleq Head(network)$ IN
   $\land\ m.kind = \text{“Ack”}$
   $\land\ m.to = v$
   $\land\ network' = Tail(network)$
   $\land\ expectedAck' = [expectedAck \text{ EXCEPT } ![v] = @ \setminus \{m.from\}]$
   $\land$ UNCHANGED $\langle route, state \rangle$

3.0.3 Équité

9. Énoncer l'équité minimale nécessaire pour que les propriétés `NoAccident` et `AbsenceDeBouchon` soient vérifiées.
   Rien de nécessaire pour `NoAccident` *(qui n'est d'ailleurs pas vérifiée).*
   Pour `AbsenceDeBouchon` *:*
   $Fairness \triangleq \forall\, v \in Vehicle :$
   $\land\ WF_{vars}(Enter(v))$
   $\land\ WF_{vars}(Leave(v))$
   $\land\ WF_{vars}(AllowRequest(v))$
   $\land\ WF_{vars}(ReceiveAck(v))$

3.1 Vérification

10. Expliquer informellement comment vérifier la propriété `NoAccident`.
    C'est un invariant : construire l'ensemble des états accessibles et vérifier le prédicat dans chaque état.
11. Montrer, sur un contre-exemple avec deux véhicules, que la propriété `NoAccident` n'est pas vérifiée. Il n'est pas nécessaire (ni même envisageable) de construire tout le graphe.

```
Request(v1, r1)
Request(v2, r3)
AllowRequest(v1) to v2
Enter(v1)
ReceiveAck(v2)
Enter(v2)
```

12. Expliquer informellement comment vérifier la propriété `AbsenceDeBouchon`.
    Propriété de vivacité

A Spécification du carrefour : `crossing.tla`

MODULE *crossing*

EXTENDS *Naturals*, *Sequences*
CONSTANTS *Vehicle*, *Route*, *Conflict*
VARIABLES *state*, *route*, *network*, *expectedAck*
$vars \triangleq \langle state, route, network, expectedAck \rangle$

$Idle \triangleq$ "Idle"
$Requesting \triangleq$ "Requesting"
$Crossing \triangleq$ "Crossing"
$NoRoute \triangleq$ "none"

Construit une séquence d'éléments à partir d'un ensemble d'éléments. L'ordre est indéterminé.
RECURSIVE $SetToSeq(\_)$
$SetToSeq(S) \triangleq$ IF $S = \{\}$ THEN $\langle\rangle$
ELSE LET $e \triangleq$ CHOOSE $x \in S$: TRUE IN $Append(SetToSeq(S \setminus \{e\}), e)$

$Init \triangleq \land route = [v \in Vehicle \mapsto NoRoute]$
$\land state = [v \in Vehicle \mapsto Idle]$
$\land network = \langle\rangle$
$\land expectedAck = [v \in Vehicle \mapsto \{\}]$

$Present \triangleq \{v \in Vehicle : state[v] \in \{Crossing, Requesting\}\}$

$Request(v, r) \triangleq$
$\land state[v] = Idle$
$\land state' = [state \text{ EXCEPT } ![v] = Requesting]$
$\land route' = [route \text{ EXCEPT } ![v] = r]$
$\land network' = network \circ SetToSeq(\{[kind \mapsto$ "Req", $from \mapsto v, to \mapsto w, route \mapsto r] : w \in Present\})$
$\land expectedAck' = [expectedAck \text{ EXCEPT } ![v] = Present]$

$AllowRequest(v) \triangleq$
$\land Len(network) > 0$
$\land$ LET $m \triangleq Head(network)$ IN
$\land m.kind =$ "Req"
$\land m.to = v$
$\land \lor state[v] \neq Crossing$
$\lor state[v] = Crossing \land \neg(m.route \in Conflict[route[v]])$
$\land network' = \langle[kind \mapsto$ "Ack", $from \mapsto v, to \mapsto m.from, route \mapsto m.route]\rangle \circ Tail(network)$
$\land$ UNCHANGED $\langle route, state, expectedAck \rangle$

$ReceiveAck(v) \triangleq TODO$

$Enter(v) \triangleq TODO$

$Leave(v) \triangleq$
$\land state[v] = Crossing$
$\land state' = [state \text{ EXCEPT } ![v] = Idle]$
$\land route' = [route \text{ EXCEPT } ![v] = NoRoute]$
$\land$ UNCHANGED $\langle network, expectedAck \rangle$

$Next \triangleq TODO$

$Spec \triangleq Init \land \Box[Next]_{vars}$

A message is a record [kind, from, to, route], where kind is "Req" or "Ack".
$TypeOk \triangleq \land route \in [Vehicle \to Route \cup \{NoRoute\}]$
$\land state \in [Vehicle \to \{Idle, Requesting, Crossing\}]$
$\land expectedAck \in [Vehicle \to \text{SUBSET } Vehicle]$
$\land network \in Seq([kind :$ {"Req", "Ack"}, $from : Vehicle, to : Vehicle, route : Route])$