Université de Sherbrooke, Département d'informatique

IGL501 : Méthodes formelles en génie logiciel, Examen final
Professeur : Marc Frappier, Lundi 16 décembre 2013, 9h00 à 12h00, local D3-2033

Documentation permise. Appareil électronique interdit. La correction est, entre autres, basée sur le fait que chacune de vos réponses soit *claire*, c'est-à-dire lisible et compréhensible pour le lecteur; *précise*, c'est-à-dire exacte et sans erreur; *concise*, c'est-à-dire qu'il n'y ait pas d'élément superflu; *complète*, c'est-à-dire que tous les éléments requis sont présents.

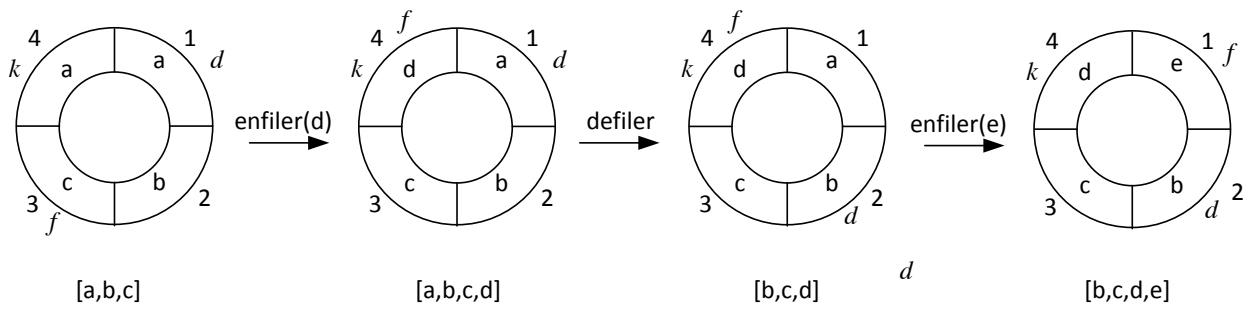
Pondération :

1	2	3	4	5	6	7	Total
30	10	30	30	10	10	10	130

1. (30 pts) Soit la spécification B suivante d'une file, qui utilise une séquence pour représenter la file. Rappel: En B, une séquence $q \in$ **seq**(*ELEM*) est une fonction de type 1..card(q) → ELEM. On peut donc utiliser pour une séquence tous les opérateurs des ensembles, fonctions et relations.

MACHINE *q1*
SETS *ELEM*
CONSTANTS *k*
PROPERTIES $k \in$ **NAT1**
VARIABLES *q*
INVARIANT $q \in$ **seq**(*ELEM*)
INITIALISATION q := []
OPERATIONS
enfiler(e) = **PRE** $e \in ELEM \wedge$ **card**(q) < k **THEN** $q := q \leftarrow e$ **END**;
defiler = **PRE card**(q) > 0 **THEN** q := **tail**(q) **END**;
$r \leftarrow$ **tete** = **PRE card**(q) > 0 **THEN** r := **first**(q) **END**
END

Complétez l'ébauche de son raffinement donné à la page suivante. Ce raffinement est basé sur le concept de *liste circulaire* pour représenter une file. La file est stockée dans le vecteur *v* de taille *k*. La variable *d* indique le début de la file (la tête). La variable *f* indique la fin (la queue) de la file. La variable *n* indique le nombre d'éléments dans la file Quand on enfile un élément, on l'ajoute à la position *f*+1. Quand on défile un élément, on incrémente simplement *d*. Si *d* ou *f* atteignent la fin du vecteur (ie, la position *k*), on continue au début du vecteur, d'où l'appellation "circulaire". Ainsi, on calcule la position suivante de k à l'aide de l'opérateur mod. Par exemple, la figure ci-dessous présente les transitions à partir d'un état où la file [a,b,c] est stockée dans le vecteur *v*, avec a en tête et c en queue.



Solution: voir q1_ref.ref

Voici quelques indices pour vous aider.

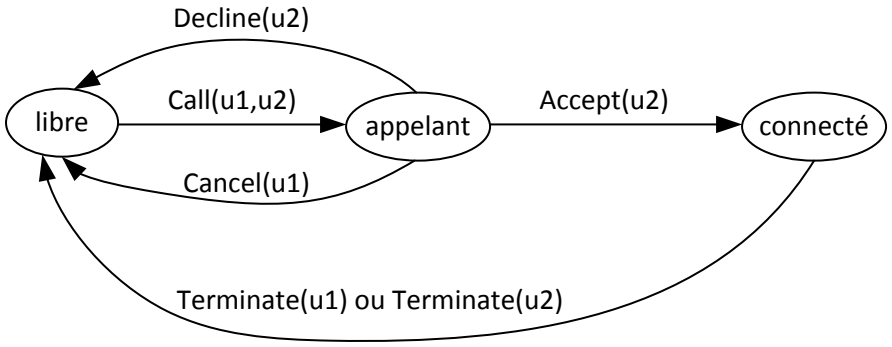
- Définissez la constante *succ_c*, une fonction qui retourne le successeur d'une position dans la liste; par exemple, *succ_c*(1) = 2 et *succ_c*(*k*) = 1. Utilisez cette fonction pour calculer les valeurs de *d* et *f*.
- Définissez la constante *qv*, une fonction qui prend en entrée *d*, le début de la file dans *v*, et *i*, une position dans la file *q*, et qui retourne la position correspondante dans *v*; par exemple,
  - *qv*(*d*)(1) = *d*
  - *qv*(*d*)(2) = *d*+1
  - *qv*(*k*)(2) = 1

REFINEMENT *q1_ref*
REFINES *q1*
CONSTANTS
succ_c, *qv*
PROPERTIES
succ_c = ...
∧ *qv* = ...
VARIABLES *v*,*d*,*f*,*n*
INVARIANT
v ∈ ... à compléter ... /* Vecteur des éléments de la file */
∧ *d* ∈ ... à compléter ... /* début : position du premier élément de la file */
∧ *f* ∈ ... à compléter ... /* fin : position du dernier élément de la file */
∧ *n* ∈ ... à compléter ... /* nb d'éléments de la file */
∧ /* invariant de collage à compléter */
... à compléter ...
INITIALISATION …
OPERATIONS
enfiler(*e*) = ... à compléter ...
defiler = ... à compléter ...
r ← **tete** = ... à compléter ...
END

2. (10 pts) Donnez le raffinement de l'opération **supprimer**(*e*), qui supprime une occurrence de l'élément *e* de la file. L'opérateur *q* ↑ *j* retourne les *j* premiers éléments de la séquence q. L'opérateur *q* ↓ *j* supprime les *j* premiers éléments de q. L'opérateur *q1* ^ *q2* retourne la concaténation des séquences *q1* et *q2*. Indice: Utilisez la fonction *succ_c* avec la composition relationnelle « ; » pour décaler facilement les éléments avant ou après *e*.

supprimer(*e*) =
PRE *e* ∈ *ELEM* **THEN**
IF *e* ∈ **ran**(*q*) **THEN**
ANY *p* **WHERE** *p* ∈ **dom**(*q*) ∧ *q*(*p*) = *e*
THEN *q* := (*q* ↑ *p*-1) ^ (*q* ↓ *p*) **END**
END
END;
Solution: voir q1_ref.ref

3. (30 pts)
(a) Spécifiez avec Alloy un système de communication sur internet similaire à Skype, et qui ressemble au système téléphonique vu en classe, mais en plus simple. Ce système se comporte comme suit. Si u1 est libre et si u2 est libre, Call(u1,u2) permet à u1 d'appeler u2; u2 peut ensuite accepter l'appel, en faisant Accept(u2), ou décliner, en faisant Decline(u2). Quand u2 accepte, u1 et u2 sont connectés et peuvent se parler. Leur conversation se termine quand l'un ou l'autre effectue un Terminate(u*x*). L'action Call(u1,u2) ne peut pas s'exécuter si u1 ou u2 n'est pas libre. L'appelant u1 peut annuler un appel qui n'est pas encore accepté ou décliné, en faisant Cancel(u1). Un utilisateur n'est pas libre quand il appelle et quand il est appelé. Il redevient libre après Decline, Cancel ou Terminate. Le diagramme état-transition ci-dessous illustre ce système. Dans l'état initial du système, on suppose que tous les utilisateurs existent et qu'ils sont libres. On ne peut pas en ajouter ou supprimer par la suite.



(b) Donnez l'invariant de ce système. Entre autres, assurez-vous d'inclure les propriétés suivantes dans votre invariant. Il y en a plusieurs autres.

- Un utilisateur ne peut appeler ou être appelé par plus d'un utilisateur à la fois.
- Un utilisateur ne peut être connecté avec plus d'un utilisateur à la fois.
- Les états libre, appelant et connecté sont mutuellement exclusifs pour un utilisateur donné (par exemple, un utilisateur ne peut être libre et connecté en même temps).

(c) Prouvez aussi la propriété suivante (qui n'est pas un invariant, donc vous devez donner un check approprié pour la prouver):

- Si deux utilisateurs sont libres, alors ils peuvent être connectés (en faisant un Call suivi d'un Accept).

Utilisez les signatures suivantes pour votre spécification. Par soucis de concision, vous n'avez pas à coder le prédicat Transition habituellement utilisé dans nos exemples (on le suppose connu!).

```
open util/ordering[State]
enum Event {init,call,accept,decline,cancel,terminate}
sig User{}
sig State
{
    libre : set User,
    appelant : User -> User,
    connecte : User -> User,
    event : Event
}
```

Solution : voir q3.als

4. (30 pts) Spécifiez en CSP le système décrit à la question précédente, en complétant le code suivant. Vous n'avez qu'à donner le code pour le processus appel.

```
nbUser = 3
USER = {1..nbUser}

-- actions du système
channel accept, decline, cancel, terminate : USER
channel call: USER.USER

-- actions internes utilisées pour le contrôle; à masquer dans le comportement final
channel liberer : USER.USER

SYNC = {| call |}
CTRL = {| liberer |}

Main = (tousAppels [| union(SYNC,CTRL) |] controleAppel({}))

MainEnv = Main \ CTRL

tousAppels = ||| n : USER @ appel(n)

controleAppel(appelant) =

    [] n1:USER, n2:USER @
            empty(inter({n1,n2},appelant)) & call!n1!n2 -> controleAppel(union(appelant,{n1,n2}))
 []
    [] n1:USER, n2:USER @ liberer!n1!n2 -> controleAppel(diff(appelant,{n1,n2}))

appel(n1) = … à compléter …
```

À titre de référence, voici une courte descriptions des fonctions sur les ensembles utilisées dans le processus controleAppel.

- union(s1,s2) : retourne l'union des ensembles s1 et s2.
- inter(s1,s2) : retourne l'intersection des ensembles s1 et s2.
- diff(s1,s2) : retourne s1 - s2.
- empty(s1) : retourne vrai ssi s1 est vide

Solution : voir q4.csp

5. (10 pts) Soit les opérations op1 et op2 suivantes.

s ← op1 = ANY x WHERE x ∈ {1,2,3} THEN s := x END

s ← op2 = CHOICE s := 1 OR s := 2 END

(a) Est-ce que op1 ⊑ op2? Donnez une preuve si le raffinement est vrai, ou un contre-exemple dans le cas contraire.

Solution : Oui

Preuve
[CHOICE s' := 1 OR s' := 2 END]¬ [ANY x WHERE x ∈ {1,2,3} THEN s := x END] (s' != s)
⇔ { définition ANY }
[CHOICE s' := 1 OR s' := 2 END]¬ (∀x. x ∈ {1,2,3} ⇒ ((s' != x))
⇔ { définition CHOICE }
¬ (∀x. x ∈ {1,2,3} ⇒ ((1 != x)) ∧ ¬ (∀x. x ∈ {1,2,3} ⇒ ((2 != x))
⇔ { ré-écriture ¬∀x en ∃x }
(∃x. x ∈ {1,2,3} ∧ ((1 = x)) ∧ (∃x. x ∈ {1,2,3} ∧ ((2 = x))
⇔ { éliminer ∃x avec x:=1 pour le premier et x:=2 pour le deuxième }
1 = 1 ∧ 2 = 2

(b) Est-ce que op2 ⊑ op1? Donnez une preuve si le raffinement est vrai, ou un contre-exemple dans le cas contraire.

Solution : Faux. Contre-exemple: la sortie s:=3 n'est pas admise au niveau abstrait.

6. (10 pts) Prouvez la transition suivante; utilisez les règles d'inférence données à la page suivante, et numérotez vos étapes de preuve.

Note: $P \underset{X}{\parallel} Q$ est noté ici $P\,|[\,X\,]|\,Q$ comme dans la version ASCII de CSP.

$$\Big(\big(a \to SKIP\,|[\,a\,]|\,(a \to STOP \,\Box\, b \to STOP)\big) \setminus \{b\}\Big)\,;\, c \to STOP \overset{a}{\longmapsto} \ldots$$

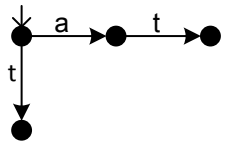
Solution

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{a \in \Sigma}{a \to SKIP} \to}{a \to SKIP \xrightarrow{a} SKIP} \quad \dfrac{\dfrac{\dfrac{a \in \Sigma}{a \to STOP} \to}{a \to STOP \xrightarrow{a} STOP}}{a \to STOP \,\Box\, b \to STOP \xrightarrow{a} STOP} \Box_3}{a \to SKIP \,|[\,a\,]|\, (a \to STOP \,\Box\, b \to STOP) \xrightarrow{a} SKIP \,|[\,a\,]|\, STOP} |[\,]|_3}{\Big(a \to SKIP \,|[\,a\,]|\, (a \to STOP \,\Box\, b \to STOP)\Big) \setminus \{b\} \xrightarrow{a} \Big(SKIP \,|[\,a\,]|\, STOP\Big) \setminus \{b\}} \setminus_1}{\Big(\Big(a \to SKIP \,|[\,a\,]|\, (a \to STOP \,\Box\, b \to STOP)\Big) \setminus \{b\}\Big) ; c \to STOP \xrightarrow{a} \Big(\Big(SKIP \,|[\,a\,]|\, STOP\Big) \setminus \{b\}\Big) ; c \to STOP} ;_1$$

7. (10 pts) Donnez le graphe de transition de l'expression suivante.

$$\Big(\Big(a \to SKIP \,|[\,a\,]|\, (a \to STOP \,\Box\, b \to STOP)\Big) \setminus \{b\}\Big) ; c \to STOP$$

Solution



Règles d'inférence de CSP

$$\frac{a \in \Sigma}{a \to P \overset{a}{\longmapsto} P}\ \to \qquad \frac{}{SKIP \overset{\checkmark}{\longmapsto} \Omega}\ SKIP$$

$$\frac{}{P \sqcap Q \overset{\tau}{\longmapsto} P}\ \sqcap_1 \qquad \frac{}{P \sqcap Q \overset{\tau}{\longmapsto} Q}\ \sqcap_2$$

$$\frac{P \overset{\tau}{\longmapsto} P'}{P \,\Box\, Q \overset{\tau}{\longmapsto} P' \,\Box\, Q}\ \Box_1 \qquad \frac{Q \overset{\tau}{\longmapsto} Q'}{P \,\Box\, Q \overset{\tau}{\longmapsto} P \,\Box\, Q'}\ \Box_2$$

$$\frac{P \overset{\sigma}{\longmapsto} P' \quad \sigma \neq \tau}{P \,\Box\, Q \overset{\sigma}{\longmapsto} P'}\ \Box_3 \qquad \frac{Q \overset{\sigma}{\longmapsto} Q' \quad \sigma \neq \tau}{P \,\Box\, Q \overset{\sigma}{\longmapsto} Q'}\ \Box_4$$

$$\frac{P \overset{\sigma}{\longmapsto} P' \quad \sigma \neq \checkmark}{P \,;\, Q \overset{\sigma}{\longmapsto} P' \,;\, Q}\ ;_1 \qquad \frac{P \overset{\checkmark}{\longmapsto} P'}{P \,;\, Q \overset{\tau}{\longmapsto} Q}\ ;_2$$

$$\frac{P \overset{\sigma}{\longmapsto} P' \quad \sigma \notin B \cup \{\checkmark\}}{P \setminus B \overset{\sigma}{\longmapsto} P' \setminus B}\ \setminus_1 \qquad \frac{P \overset{\checkmark}{\longmapsto} P'}{P \setminus B \overset{\checkmark}{\longmapsto} \Omega}\ \setminus_2 \qquad \frac{P \overset{\sigma}{\longmapsto} P' \quad \sigma \in B}{P \setminus B \overset{\tau}{\longmapsto} P' \setminus B}\ \setminus_3$$

$$\frac{P \overset{\sigma}{\longmapsto} P' \quad \sigma \notin X \cup \{\checkmark\}}{P \,|[\, X \,]|\, Q \overset{\sigma}{\longmapsto} P' \,|[\, X \,]|\, Q}\ |[\,]|_1 \qquad \frac{Q \overset{\sigma}{\longmapsto} Q' \quad \sigma \notin X \cup \{\checkmark\}}{P \,|[\, X \,]|\, Q \overset{\sigma}{\longmapsto} P \,|[\, X \,]|\, Q'}\ |[\,]|_2$$

$$\frac{P \overset{\sigma}{\longmapsto} P' \quad Q \overset{\sigma}{\longmapsto} Q' \quad \sigma \in X}{P \,|[\, X \,]|\, Q \overset{\sigma}{\longmapsto} P' \,|[\, X \,]|\, Q'}\ |[\,]|_3$$

$$\frac{P \overset{\checkmark}{\longmapsto} P'}{P \,|[\, X \,]|\, Q \overset{\tau}{\longmapsto} \Omega \,|[\, X \,]|\, Q}\ |[\,]|_4 \qquad \frac{Q \overset{\checkmark}{\longmapsto} Q'}{P \,|[\, X \,]|\, Q \overset{\tau}{\longmapsto} P \,|[\, X \,]|\, \Omega}\ |[\,]|_5$$

$$\frac{}{\Omega \,|[\, X \,]|\, \Omega \overset{\checkmark}{\longmapsto} \Omega}\ |[\,]|_6$$

Fin de l'examen