Université de Sherbrooke
Département d'informatique

IGL501-IGL710 : Méthodes formelles en génie logiciel

Examen périodique

Professeur : Marc Frappier

Samedi 19 octobre 2019, 9 h 00 à 12 h 00.

Notes importantes :

- Toute documentation permise.
- 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.
- nombre de pages de l'examen, incluant celle-ci : 4.

Pondération :

Question	Point	Résultat
1	20	
2	20	
3	35	
4	25	
total	100	

Nom : _____ Prénom : _____

Signature : _____ Matricule : _____

1. (**20 pt**) Traduisez les énoncés suivants avec le langage de Tarski.

   (a) Il existe un carré à la gauche de tous les triangles.
   Solution:
   $\exists x(\mathsf{Square}(x) \wedge \forall y(\mathsf{Triangle}(y) \Rightarrow \mathsf{LeftOf}(x, y)))$

   (b) Si deux carrés sont sur la même ligne, alors ils sont de même taille.
   Solution:
   $\forall x \forall y(\mathsf{Square}(x) \wedge \mathsf{Square}(y) \wedge \mathsf{SameRow}(x, y) \Rightarrow \mathsf{SameSize}(x, y))$

   (c) Une condition suffisante pour que les carrés soient petits est que les pentagones soient petits.
   Solution:
   $\forall x(\mathsf{Pentagon}(x) \Rightarrow \mathsf{Small}(x)) \Rightarrow \forall x(\mathsf{Square}(x) \Rightarrow \mathsf{Small}(x))$

   (d) Une condition nécessaire pour que pentagones soient petits est que les carrés soient petits.
   Solution:
   $\forall x(\mathsf{Pentagon}(x) \Rightarrow \mathsf{Small}(x)) \Rightarrow \forall x(\mathsf{Square}(x) \Rightarrow \mathsf{Small}(x))$

   (e) L'objet le plus grand est un carré.
   Solution:
   $\exists x(\mathsf{Square}(x) \wedge \forall y(x \neq y \Rightarrow \mathsf{Smaller}(y, x))$

2. (**20 pt**) Pour chaque opération suivante, indiquez si elle préserve l'invariant. Si elle le préserve, justifiez votre réponse (un texte suffit; vous pouvez aussi donnez une preuve si vous préférez). Si elle ne le préserve pas, donnez un contre-exemple et trouvez la précondition la plus faible (la moins restrictive) qui permet de préserver l'invariant.

   (a) Opération : `A(x) = PRE x : 0..3 & y : 3..k THEN y := y-x END`
   Invariant : `y : 0..k`
   Solution: Oui

   (b) Opération : `B(x) = PRE x : 1..2 & y : -4..3 THEN CHOICE y:=y+x OR y:=y-x END END`
   Invariant : `y : -5..5`
   Solution: Non. Contre-exemple: $x = 2$ et $y = -4$ avec $y := y - x$. Précondition: $y : -3..3$

   (c) Opération : `C(x,y) = PRE x : 0..k & y 0..k THEN f(x) := f(y) || f(y) := f(x) END`
   Invariant : `f : 0..k >->> 0..k`
   Solution: Oui

   (d) Opération : `D = ANY x,y WHERE x : 0..3 & y : 0..3 THEN z := z+x+y END`
   Invariant : `z : 0..9`
   Solution: Non. Contre-exemple: $x = 3$ et $y = 3$ avec $z : 4..MAXINT$. Précondition: $z < 4$

   (e) Opération : `D = SELECT x > 1 THEN x := x-1 WHEN x < 1 THEN x := -x END`
   Invariant : `x : NAT`
   Solution: Oui

3. (**35 pt**) Modélisez une liste de nombres naturels en B. Un nombre ne peut apparaitre plus d'une fois dans la liste. Les positions des éléments de la liste débutent à 1. La liste a une capacité maximale de k. Voici les opérations à spécifier.

   - add(x : NAT)
     Ajoute l'élément x à la fin de la liste. L'élément ne doit pas déjà appartenir à la liste.
   - addp(x : NAT, p : NAT)
     Ajoute l'élément x à la position p de la liste. L'élément ne doit pas déjà appartenir à la liste. Les éléments qui sont dans la liste à partir de la position p sont décalés d'une place, ie, si $p' \geq p$ est la position d'un élément, alors il se retrouve en position $p' + 1$ après l'insertion de x. Soit n la taille de la liste. Si $p = n + 1$, alors l'élément est ajouté à la fin de la liste.
   - del(x : NAT)
     Supprime l'élément x de la liste si il est présent. Si x n'est pas présent, alors la liste est inchangée.
   - del(p : NAT)
     Supprime l'élément à la position p. La position doit exister dans la liste.
   - sort
     Trie les éléments de la liste en ordre croissant.

4. (**25 pt**) Modélisez en B le système suivant. Un ensemble d'utilisateurs U veulent accéder à un ensemble de ressources R. Un seul utilisateur peut utiliser une ressource à la fois. Si plusieurs utilisateurs veulent la même ressource, alors les demandes sont servies selon l'ordre d'arrivée; les demandes sont donc traitées comme une file d'attente. Spécifier les opérations suivantes.

   - demander(u : U, r : R)
     L'utilisateur u demande la ressource r.
   - allouer(r : R)
     Le système alloue la ressource r à l'utilisateur qui est en tête de la file d'attente de la ressource.
   - libérer(r : R)
     L'utilisateur libère la ressource r.

Description	**Expression**	**Syntaxe ASCII B**	**Définition/Exemple**
suite vide	$[\,]$	`[]`	
suite par extension	$[t_1, \ldots, t_n]$	`[`$t_1, \ldots, t_n$`]`	
suite sur S	$\mathsf{seq}(S)$	`seq`(S)	$\{f \mid f \in \mathbb{N} \pfun S \land \mathsf{finite}(f) \land \mathsf{dom}(f) = 1..\mathsf{card}(f)\}$
suite non-vide sur S	$\mathrm{seq}_1(S)$	`seq1`(S)	$\mathsf{seq}(S) - [\,]$
suite injective sur S	$\mathsf{iseq}(S)$	`iseq`(S)	$\mathsf{seq}(S) \cap \mathbb{N} \pinj S$
suite inj. non-vide sur S	$\mathrm{iseq}_1(S)$	`iseq1`(S)	$\mathrm{iseq}_1(S) - [\,]$
concaténation	$s_1 \frown s_2$	s_1`^`s_2	$[a,b] \frown [c,d] = [a,b,c,d]$
premier élément	$\mathsf{first}(s)$	`first`(s)	$s \neq [\,]$, $\mathsf{first}([a,b,c]) = a$
sauf premier élément	$\mathsf{tail}(s)$	`tail`(s)	$s \neq [\,]$, $\mathsf{tail}([a,b,c]) = [b,c]$
dernier élément	$\mathsf{last}(s)$	`last`(s)	$s \neq [\,]$, $\mathsf{last}([a,b,c]) = c$
sauf dernier élément	$\mathsf{front}(s)$	`front`(s)	$s \neq [\,]$, $\mathsf{front}([a,b,c]) = [a,b]$
inverse	$\mathsf{rev}(s)$	`rev`(s)	$\mathsf{rev}([a,b,c]) = [c,b,a]$
ajout de e au début de s	$e \rightarrow s$	`e -> s`	$c \rightarrow [a,b] = [c,a,b]$
ajout de e à la fin de s	$s \leftarrow e$	`s <- e`	$[a,b] \leftarrow c = [a,b,c]$

Table 1: Opérations et prédicats sur les suites

Fin de l'examen