# Université de Sherbrooke, Département d’informatique

IGL501/IGL710 : Méthodes formelles en génie logiciel, Examen final

Professeur : Marc Frappier, 18 décembre 2019, 9 h à 12 h

Salles : D7-2023

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. Cet examen comporte 4 pages, incluant celle-ci.

Pondération :

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Question** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | Total |
| **Valeur** | 15 | 15 | 15 | 15 | 15 | 15 | 10 | 100 |

1. (15 pts) Soit la machine B ci-dessous.

MACHINE Q1

VARIABLES S

INVARIANT S ⊆ NAT

INITIALISATION S := ∅

OPERATIONS

 ajout(x) = PRE x ∈ NAT ∧ x ∉ S

 THEN S := S ∪ {x} END

;

 sup(x) = PRE x ∈ NAT ∧ x ∈ S

 THEN S := S - {x} END

;

 r <-- maximum =

 IF S ≠ ∅ THEN r := max(S) ELSE r := 0 END

END

Spécifiez un raffinement qui utilise comme variables d’état S1 et S2 telles que

* S1 ⊆ NAT et S2 ⊆ NAT
* S1 ne contient que les nombres pairs de S
* S2 ne contient que les nombres impairs de S.

**Solution**

REFINEMENT Q1\_ref

REFINES Q1

VARIABLES S1,S2

INVARIANT

 S1 ⊆ NAT

∧ S2 ⊆ NAT

∧ ∀x.(x ∈ S1 ⇒ x mod 2 = 0)

∧ ∀x.(x ∈ S2 ⇒ x mod 2 ≠ 0)

∧ S = S1 ∪ S2

INITIALISATION

 S1 := ∅

|| S2 := ∅

OPERATIONS

 ajout(x) =

 PRE x ∈ NAT ∧ x ∉ S1 ∪ S2

 THEN IF x mod 2 = 0 THEN S1 := S1 ∪ {x}

 ELSE S2 := S2 ∪ {x} END

 END

;

 sup(x) =

 PRE x ∈ NAT ∧ x ∈ S1 ∪ S2

 THEN IF x mod 2 = 0 THEN S1 := S1 - {x}

 ELSE S2 := S2 - {x} END

 END

;

 r <-- maximum = IF S1∪S2 ≠ ∅ THEN r := max(S1∪S2) ELSE r := 0 END

END

2. (15 pts) Écrivez une substitution P en B0 (ie, le langage d’implémentation de B) telle que

 [P](inverse(v,w)).

Le prédicat inverse(v,w) est définit comme suit. Soit k ∈ NAT, S = 1..k, v ∈ S → S et w ∈ S → S.

inverse(v,w) ⇔ ∀x.(x ∈ S ⇒ v(k+1-x) = w(x))

Autrement dit, écrivez un programme P tel que P mets dans le vecteur w l’inverse du vecteur v. Par exemple, si

 v = {(1,4),(2,3),(3,2),(4,1)} alors w = {(1,1),(2,2),(3,3),(4,4)}

Le vecteur v est inchangé par l’exécution de P. Donnez l’invariant et le variant de la boucle WHILE.

**Solution**

VAR i IN

i := 0;

WHILE (i < k) DO

 i := i+1;

 w(i) := v(k+1-i)

INVARIANT

 !x.(x : 1..i => w(x) = v(k+1-x))

& i : 0..k

VARIANT

 (k-i)

END

END

3. (15 pts) Soit

 P = ((a -> SKIP ||| b -> SKIP) ; c-> STOP ) [] d -> STOP

1. Prouvez la transition P -a-> P’

**Solution**

---------------------------------------------------- ⟶

a -> SKIP

-a->

SKIP

---------------------------------------------------- [||]1

a -> SKIP ||| b -> SKIP

-a->

SKIP ||| b -> SKIP

---------------------------------------------------- ;1

 (a -> SKIP ||| b -> SKIP) ; c-> STOP

-a->

(SKIP ||| b -> SKIP) ; c-> STOP

---------------------------------------------------- []3

((a -> SKIP ||| b -> SKIP) ; c-> STOP ) [] d -> STOP

-a->

((SKIP ||| b -> SKIP) ; c-> STOP )

1. Donnez l’arbre de transition de P

Solution



4. (15 pts) Soit

P1 = ((a -> STOP) [] (a -> STOP)) |~| (d -> STOP)

P2 = (a -> STOP) [] (d -> STOP)

P3 = (a -> STOP) |~| (d -> STOP)

Indiquez pour chacune des assertions suivantes si elle est vraie ou fausse; justifiez brièvement votre réponse.

1. assert P1 [T= P2
2. assert P1 [T= P3
3. assert P1 [F= P2
4. assert P1 [F= P3

**Solution**

assert P1 [T= P2 -- TRUE

assert P1 [T= P3 -- TRUE

assert P1 [F= P2 -- TRUE

assert P1 [F= P3 -- TRUE

5. (15 pts) Cette question porte sur la modélisation en Alloy de l’héritage dans les langages de programmation orientés objets comme Java ou C++, en utilisant les signatures suivantes :

|  |  |
| --- | --- |
| sig Classe {ext : lone Classe,methodes : Methode -> Corps } | sig Methode {}sig Corps {}sig Objet {type : Classe } |

La signature Classe représente les classes d’un programme C++ ou Java. Son attribut ext indique la classe dont une classe hérite. Pour simplifier, une classe peut hériter d’une seule autre classe. L’attribut methodes indique les méthodes de la classe, avec leur corps. Une méthode peut apparaitre dans plusieurs classes, avec des corps différents. Le type d’un objet est une classe, tel qu’indiqué par l’attribut type. Voici un exemple de valeurs dans ce modèle.

|  |  |
| --- | --- |
| one sig C1,C2,C3 extends Classe {}one sig M1,M2,M3 extends Methode {}one sig B1,B2,B3,B3,B4,B5 extends Corps {}one sig O1 extends Objet {} | fact {ext = C3 -> C2 + C2 -> C1methodes = C1 -> M1 -> B1+ C1 -> M2 -> B2+ C2 -> M1 -> B3+ C2 -> M2 -> B4+ C3 -> M1 -> B5type = O1 -> C3} |

Dans cet exemple, la classe C2 hérite de la classe C1. On dit alors que C2 est plus spécifique que C1. L’héritage est transitif, donc C3 hérite aussi de C1 par transitivité. La méthode M1 est définie dans les trois classes, chacune avec un corps différent.

Questions

1. Spécifiez un fact qui indique que dans une classe, une méthode est associée à un seul corps.

**Solution**

fact {

all c : Classe | c.methodes in Methode lone -> lone Corps

}

1. Spécifiez le prédicat suivant

pred herite[o:Objet,c:Classe,m:Methode]

qui retourne vrai ssi l’objet o *hérite* de la méthode m de la classe c. Cela est défini comme suit. Soit c' le type de o. L’objet o hérite d’une méthode m si elle est déclarée dans c' ou bien dans une classe c tell que c' hérite de c.

**Solution**

pred herite[o:Objet,c:Classe,m:Methode]

{

 c in o.type.\*ext

and m in c.methodes.Corps

}

1. Spécifiez le prédicat suivant

pred plusSpecifique[c,c':Classe]

qui retourne vrai ssi c est plus spécifique que c' (ie, c hérite, directement ou indirectement, de c’).

**Solution**

pred plusSpecifique[c,c':Classe]

{

c' in c.^ext

}

1. Spécifiez la fonction suivante

fun executer[o : Objet, m : Methode]: Corps

qui retourne le corps de la méthode qui sera exécutée si on applique la méthode m à l’objet o. La méthode exécutée provient de la classe la plus spécifique de laquelle o hérite. Si o n’hérite pas de la méthode m, alors la fonction retourne un ensemble vide. Voici quelques exemples d’appel

executer[O1,M1] = {B5}

executer[O1,M2] = {B4}

executer[O1,M3] = {}

**Solution**

fun executer[o : Objet, m : Methode]: Corps

{

 { b : Corps |

 some c : Classe | herite[o,c,m] and b = m.(c.methodes) and

 no c' : Classe | herite[o,c',m] and plusSpecifique[c',c]

 }

}

6. (15 pts) Soit la spécification B suivante

|  |  |
| --- | --- |
| MACHINE relationSETS ELEM={a,b,c}VARIABLES rINVARIANT r ∈ ELEM ↔ ELEM ∧ id(ELEM) ∩ r = ∅INITIALISATION r := ∅ | OPERATIONS add(x,y) = PRE x ∈ ELEM ∧ y ∈ ELEM ∧ (x,y) ∉ r ∧ x ≠ y THEN r := r ∪ {(x,y)} END; del(x,y) = PRE x ∈ ELEM ∧ y ∈ ELEM ∧ (x,y) ∈ r THEN r := r - {(x,y)} ENDEND |

1. Traduisez cette spécification B en Alloy.

**Solution**

open util/ordering[State]

enum Event {init,add, del}

enum ELEM {a,b,c}

sig State {

 r : ELEM -> ELEM,

 event : Event

}

pred add[s,s' : State, x,y : ELEM]

{

x != y

x->y not in s.r

s'.r = s.r + x->y

s'.event = add

}

pred del[s,s' : State, x,y : ELEM]

{

x->y in s.r

s'.r = s.r - x->y

s'.event = del

}

pred Init[s:State]

{

no s.r

s.event = init

}

1. Donnez une commande qui permet de prouver l’invariant de la machine B.

**Solution**

pred inv[s : State]

{

no iden & s.r

}

check Inv {

all s : State | Init[s] => inv[s]

all s,s' : State | inv[s] and Transition[s,s'] => inv[s']

} for 2 State

1. Soit x,y : ELEM, s1,s2,s3 : State. Donnez une commande qui vérifie que tout appel de add[s1,s2,x,y] suivi de del[s2,s3,x,y] laisse la relation inchangée.

**Solution**

check add\_del {

 all x,y : ELEM, s1, s2, s3 : State |

 add[s1,s2,x,y]

 and add[s2,s3,x,y]

 =>

 s1.r = s3.r

} for 3 State

7. (15 pts) Soit la spécification CSP suivante.

channel a,b,c

P1 = a -> ((b -> P2) [] (c -> P3))

P2 = (b -> P2) [] (b -> P1)

P3 = (c -> P3) [] (c -> P1)

Écrivez une formule de logique temporelle (LTL ou CTL, à votre choix) pour chacune des propriétés suivantes.

1. Un a est toujours suivi d’un b ou d’un c

**Solution**

G ([a] => F([b] or [c]))

Ou bien

G ([a] => X([b] or [c]))

1. Si un b survient, alors aucun c ne survient avant l’arrivée d’un a

**Solution**

G ([b]=> ((not [c]) W [a]))

1. Il existe une trace dont le suffixe est une suite infinie de b. Autrement dit, la fin de la trace est une suite infinie de b.

**Solution**

Il suffit que la formule LTL suivante soit fausse

not (FG [b])

ou bien que la formule CTL suivante soit vraie

EF EG [b]

Fin de l'examen; Joyeuses Fêtes!