EFREI Master 1

Correction DE (SCV)

Réseaux de Petri

Les deux programmes suivants coopèrent dans le but d'assurer l'exclusion mutuelle à leurs sections critiques.

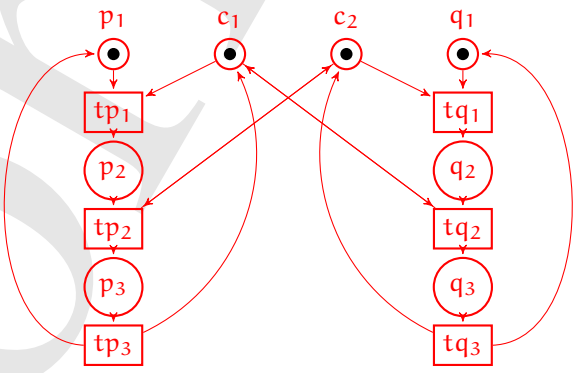
```
boolean c1 , c2 = true ;
process P1( )
{
  while (true)
  {
     //non critique 1 ;
     c1 = fa l s e ;
     while(c2 == false);
     //critique1;
     c1=true;
   }
 }
```

```
process P2( )
{
  while (true)
  {
    //non critique 2 ;
     c2 =false;
     while(c1==false) ;
     //critique 2;
     c2=true;
   }
 }
```

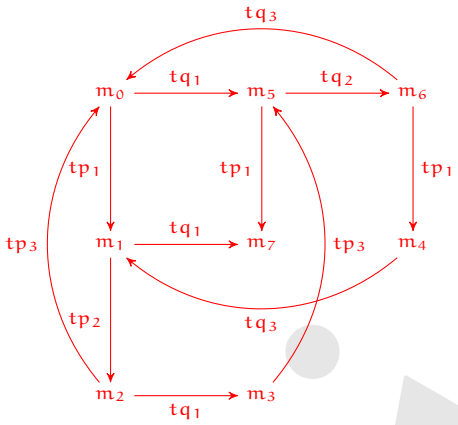
1. Modélisez avec des réseaux de Petri le programme ci-dessus. Utiliser des places pour modéliser les points de contrôle des processus et les états des variables partagées.
2. Exprimez la propriété d'exclusion mutuelle entre les deux processus ainsi que la propriété d'absence de blocage, en utilisant les transitions/places du réseau, avec une logique temporelle.
3. En utilisant le système de transition correspondant (graphe des marquages accessibles), vérifiez les deux propriétés mentionnées ci-dessus.

Correction :

1.



2. Expression des propriétés d'exclusion mutuelle et d'absence de blocage en logique LTL :
   — exclusion mutuelle $G!(p_3 \wedge q_3)$
   — absence de blocage $GX\ true$



3. $m_0 = p_1 + c_1 + c_2 + q_1$
   $m_1 = p_2 + c_2 + q_1$
   $m_2 = p_3 + c_2 + q_1$

   $m_3 = p_3 + q_2$
   $m_4 = p_3 + q_2$
   $m_5 = p_1 + c_1 + q_2$
   $m_6 = p_1 + c_1 + q_3$
   $m_7 = p_2 + q_2$

   — Graphe des marquages accessibles :
   — Exclusion mutuelle : vérifiée (dans aucun marquage p_3 et q_3 sont toutes les deux marquées
   — Absence de blocage : non vérifiée (le marquage m_7 est un marquage mort

◊

LTL et Automates de Büchi

1. Exprimer en LTL les propriétés suivantes :
   (a) à l'instant suivant, si p est vrai alors q n'est jamais vrai
   (b) p sera vrai au plus une fois
   (c) p sera vrai exactement 2 fois.

Correction :

(a) à l'instant suivant, si p est vrai alors q n'est jamais vrai : $X(p \implies G\neg q)$ ou encore $((Xp) \implies (XG\neg q))$ (la formulation est un peu ambiguë, on ne sait pas trop si q doit être faux après p ou à partir de p, ici je choisis à partir de p)

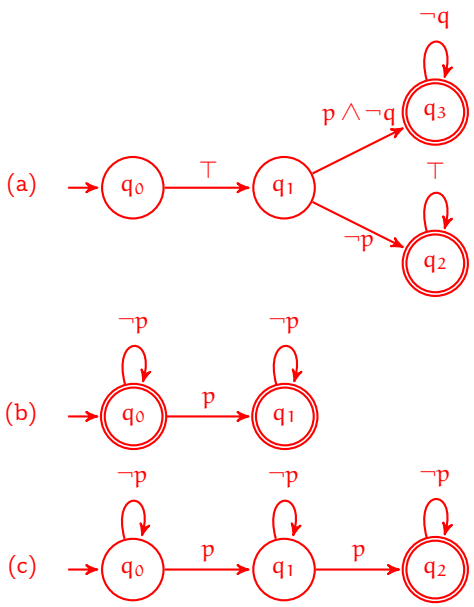
(b) p sera vrai au plus une fois : $G\neg p \vee G(p \implies X(G\neg p))$ ou $((G\neg p) \vee (\neg pU(p \wedge X(G\neg p))))$

(c) p sera vrai exactement 2 fois : $(\neg pU(p \wedge X(\neg pU(p \wedge X(G\neg p)))))$

◊

2. Pour chaque formule ci-dessus, dessinez l'automate de Büchi correspondant.

Correction :



$\Diamond$

3. Comparez les formules suivantes et justifiez vos réponses :
   (a) $G(Fp \wedge Fq)$ et $GFp \wedge GFq$.
   (b) $F(Gp \wedge Gq)$ et $FGp \wedge FGq$.
   (c) $G(Fp \vee Fq)$ et $GFp \vee GFq$.
   (d) $F(Gp \vee Gq)$ et $FGp \vee FGq$.
   (e) $GF(p \wedge q)$ et $GFp \wedge GFq$.
   (f) $FG(p \wedge q)$ et $FGp \wedge FGq$.
   (g) $GF(p \vee q)$ et $GFp \vee GFq$.
   (h) $FG(p \vee q)$ et $FGp \vee FGq$.
   (i) $q \mathcal{U} Fp$ et Fp.
   (j) $Gq \vee G(\neg p)$ et $(Fp) \implies Gq$.

Correction :

(a) $\equiv$
(b) $\equiv$
(c) $\equiv$
(d) $\equiv$
(e) $\not\equiv$
(f) $\equiv$
(g) $\equiv$
(h) $\equiv$
(i) $\equiv$
(j) $\equiv$

$\Diamond$

4. Considérons une formule LTL du type $\alpha \wedge \beta$ où α et β sont deux sous-formules LTL. Si l'on sait montrer que $\alpha \implies \beta$ alors la formule $\alpha \wedge \beta$ peut âtre réécrite simplement α. Par exemple la formule $(G\alpha) \wedge (X\alpha)$ peut-être simplifiée en $G\alpha$, car on a $(G\alpha) \implies (X\alpha)$.
   (a) Supposez que l'on vous demande d'écrire un algorithme de simplification de formules LTL implémentant ce type de réécritures. Étant données deux sous-formules α et β, comment feriez-vous pour tester si $\alpha \implies \beta$?
   Indice : pensez à ce que cela signifie au niveau des langages, et comment cela peut-être testé avec des automates.)
   (b) Supposez que vous ayez prouvé que $\alpha \implies \beta$. Comment simplifieriez-vous la formule LTL $\alpha U \beta$?
   (c) Quelle condition sur α et β faut-il pour avoir le droit de simplifier $\alpha U \beta$ en $F\beta$?

Correction :

1. : $\alpha \implies \beta$ sis $L(A_\alpha \times L_{\neg\beta}) = \emptyset$. On peut donc traduire α et $\neg\beta$ sous forme d'automates et vérifier que leur produit est vide.
2. Si $\alpha \implies \beta$ alor $\alpha \cup \beta \equiv \beta$
3. Si $\neg\beta \implies \alpha$ alors $\alpha \cup \beta \equiv F\beta$

$\Diamond$

OCL

1. Le directeur d'une chaîne d'hôtels vous demande de concevoir une application de gestion de ses hôtels. Un hôtel est constitué d'un certain nombre de chambres. Un responsable de l'hôtel gère la location des chambres. Chaque chambre se loue à un prix donné. L'accès aux salles de bains est compris dans le prix de la location d'une chambre. Certaines chambres comportent une salle de bains, mais pas toutes. Les hôtes de chambres sans salle de bain peuvent utiliser une salle de bains sur le palier. Ces dernières peuvent être utilisées par plusieurs hôtes. Les pièces de l'hôtel qui ne sont ni des chambres ni des salles de bain (hall d'accueil, cuisine...) ne font pas partie de l'étude (hors sujet). Des personnes peuvent louer une ou plusieurs chambres d'hôtel afin d'y résider. En d'autres termes : l'hôtel héberge un certain nombre de personnes, ses hôtes (il s'agit des personnes qui louent au moins une chambre de l'hôtel). Le diagramme de classes de la Figure 1 modélise ce problème :
   Donnez une formulation en langage naturel pour chacune des contraintes OCL suivantes :
   (a)
```
context Chambre inv :
  self.etage <>13
context SalleDeBains inv :
  self.etage <>13
```
   (b)
```
context Chambre inv :
 client->size <= nombreDeLits or
   (client->size = nombreDeLits +1 and
     client->exists(p:Personne | p.age < 4))
```
   (c)
```
context Hotel inv :
  Sequence{etageMin..etageMax}->forAll(i : Integer |
    if i<>13 then
      self.chambre->select(c : Chambre | c.etage = i)->notEmpty
endif
```

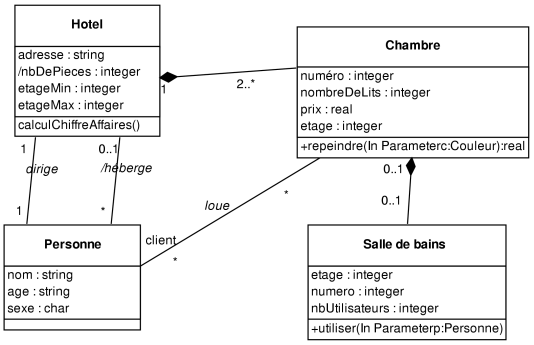


FIGURE 1 – Diagramme de classes : gestion d'hôtels

(d)
```
context Chambre::repeindre(c:Couleur)
    pre :  client->isEmpty
    post : prix = prix@pre * 1.1
```

(e)
```
context SalleDeBains::utiliser(p:Personne)
    pre : if chambre->notEmpty then
            chambre.client->includes(p)
          else
            p.chambre.etage = self.etage
          endif
    post : nbUtilisateurs = nbUtilisateurs@pre + 1
```

Correction :

(a) Un hôtel ne contient jamais d'étage numéro 13

(b) Le nombre de personnes par chambre doit être inférieur ou égal au nombre de lits dans la chambre louée. Les enfants (accompagnés) de moins de 4 ans ne comptent pas dans cette règle de calcul (à hauteur d'un enfant de moins de 4 ans maximum par chambre).

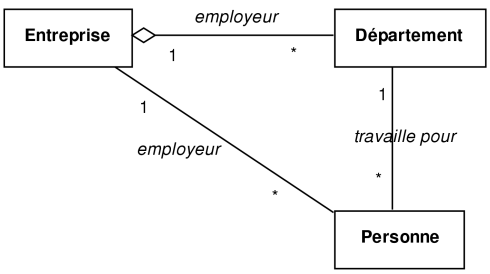
(c) Chaque étage possède au moins une chambre (sauf le 13 qui n'existe pas, bien entendu ...)

(d) On ne peut repeindre une chambre que si elle n'est pas louée. Une fois repeinte, une chambre coûte 10% de plus.

(e) Une salle de bain privative ne peut être utilisée que par des personnes qui louent la chambre contenant la salle de bains et une salle de bains sur le pallier ne peut être utilisée que par les clients qui logent sur le même palier

◊

2. Considérez le diagramme de classes suivant :



(a) Donnez une expression OCL qui permette d'indiquer que la personne qui travaille dans le département est la même que celle qui est employée par l'entreprise.

(b) Question : Donner une expression OCL qui permette d'indiquer qu'une personne travaillant pour une entreprise doit être âgée de 18 ans et plus. On suppose que la classe Personne a un attribut âge.

(c) Donner une expression OCL qui permette d'indiquer que deux personnes ne doivent pas avoir le même nom.

(d) Les personnes qui travaillent dans l'entreprise sont âgées de 18 à 65 ans. Donner l'expression OCL correspondante.

Correction :

1.
```
context Personne inv:
self.employeur = self.departement.employeur
```
2.
```
context Personne inv :
self.age > 18
```
3.
```
context Personne inv:
Personne.allinstances -> forall (p1, p2 | p1<> p2 implies p1.nom <> p2.nom)
```
4.
```
context Entreprise inv :
self.employeur.forall(Personne p | p.age<= 18 and p.age <= 65)
```

◊