Université de Sherbrooke, Département d'informatique

IGL501 : Méthodes formelles en génie logiciel, Examen final
Professeur : Marc Frappier, Lundi 16 décembre 2014, 9h00 à 12h00, local D3-2033

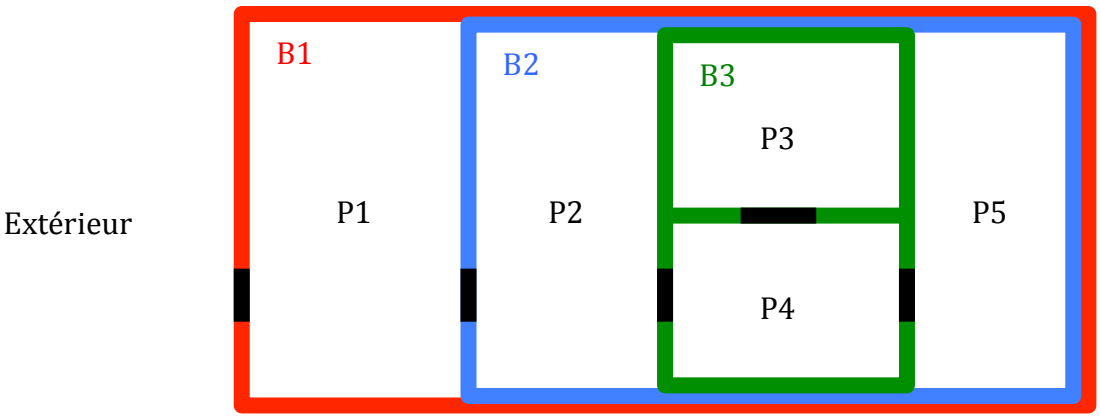
Documentation permise. Appareil électronique interdit. La correction est, entre autres, basée sur le fait que chacune de vos réponses soit *claire*, c'est-à-dire lisible et compréhensible pour le lecteur; *précise*, c'est-à-dire exacte et sans erreur; *concise*, c'est-à-dire qu'il n'y ait pas d'élément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

Pondération :

1	2	3	4	Total
50	30	20	20	120

1. (50 pts) Spécifiez en Alloy un système de contrôle d'accès dans un bâtiment. Voici les contraintes de ce problème.

- Un bâtiment est composé de pièces.
- Pour simplifier la spécification du contrôle d'accès, on considère aussi qu'un bâtiment peut aussi être composé de bâtiments. Par exemple, le bâtiment ci-dessous est composé de 5 pièces. Le bâtiment en rouge (B1) comprend la pièce P1 et le bâtiment B2. B2 comprend les pièces P2 et P5, ainsi que le bâtiment B3. B3 comprend les pièces P3 et P4.
- L'extérieur du bâtiment principal est considéré comme une pièce spéciale, afin de généraliser la notion de déplacement vers le bâtiment et à l'intérieur du bâtiment.
- Les pièces sont reliées par des portes, indiquées en noir sur le diagramme. Une porte ne relie pas des bâtiments (puisque les bâtiments sont des regroupements de pièces ou de bâtiments).



a) Donnez les signatures en Alloy pour représenter cette structure.

b) Donnez les `fact` nécessaires pour représenter les contraintes suivantes

1) Un bâtiment ne peut se contenir lui-même, directement ou indirectement.
2) Un bâtiment doit comprendre au moins un autre objet (une pièce ou un autre bâtiment).
3) Une pièce ne comprend rien.
4) Chaque pièce est dans un seul bâtiment, sauf l'extérieur (rappel, l'extérieur est une pièce spéciale).
5) L'extérieur n'est dans aucun bâtiment.
6) Chaque pièce est accessible de l'extérieur, en passant à travers une ou plusieurs portes.

7) Il y a un seul bâtiment principal, qui contient tous les autres bâtiments ou pièces (ex: B1 est le bâtiment principal dans la figure ci-dessus).

c) Spécifiez les opérations suivantes et la signature de l'état nécessaire pour spécifier ces opérations. Ces opérations permettent d'autoriser une personne à accéder à un bâtiment ou à une pièce, de révoquer une autorisation attribuée, et de permettre à une personne de se déplacer vers une pièce. Le système conserve la position d'une personne. L'autorisation d'accéder à un bâtiment donne le droit d'accéder à toutes les pièces de ce bâtiment et des bâtiments qu'il comprend. Par exemple, si une personne est autorisée à accéder à B1 selon la figure ci-dessus, alors elle peut alors accéder à toutes les pièces (P1 à P5). Naturellement, une personne est toujours autorisée à être à l'extérieur du bâtiment.

1) `pred Init [E:Etat]` /* dénote l'état initial du système */
   - Aucune autorisation, sauf celle permettant à chaque personne d'être à l'extérieur.
   - Chaque personne est à l'extérieur du bâtiment
2) `pred Autoriser[p:Personne, b:Batiment,E,E':Etat]`
   - La personne n'est pas déjà autorisée à ce bâtiment.
3) `pred Revoquer[p:Personne, b:Batiment,E,E':Etat]`
   - La personne est autorisée à ce bâtiment.
   - On ne peut révoquer d'être à l'extérieur du bâtiment
   - On ne peut révoquer une autorisation qui empêcherait une personne de sortir du bâtiment (pour des raisons de sécurité en cas de sortie d'urgence). Par exemple, si une personne est dans la pièce P2, on ne peut révoquer son autorisation d'être dans la pièce P1, car cela l'empêcherait de sortir du bâtiment.
4) `pred Deplacer[pe:Personne, pi:Piece,E,E':Etat]`
   - La personne se déplace vers la pièce `pi`.
   - La pièce `pi` doit être accessible par une porte entre sa pièce actuelle et la pièce `pi`.
   - La personne doit être autorisée à la pièce `pi`.

d) Spécifiez les invariants suivants. Pour simplifier, ne donnez pas les commandes `run` ou `check`, ni le prédicat `TraceValide`. Donnez seulement le prédicat de chaque invariant.

1) Chaque personne est dans une pièce (une personne ne peut être nul part).
2) Une personne ne peut être que dans une seule pièce à la fois.
3) Si une personne est dans une pièce seulement si elle y est autorisée.

Solution

```
open util/ordering[Etat]

enum Evenement {init, autoriser, revoquer, deplacer}

sig Batiment {
estDans : lone Batiment}

sig Piece extends Batiment {
porte : set Piece
}

fact {
porte = ~porte
```

```
no (porte & iden)
}

one sig Exterieur extends Piece {}

fact {
-- pas de cycle dans estDans
no iden & ^estDans
-- pas de batiment sans piece
all b:Batiment-Piece | some estDans.b
-- chaque piece, sauf extérieur, est dans un batiment
(Piece-Exterieur) in estDans.Batiment
-- Extérieur est dans aucun batiment
Exterieur not in estDans.Batiment
-- Aucun batiment dans une pièce
no estDans.Piece
-- chaque piece est accessible de l'extérieur
Exterieur->(Piece-Exterieur) in ^porte
-- Un seul bâtiment racine
#(Batiment-estDans.Batiment)=2
}

sig Personne {}

sig Etat {
autorisation : Personne -> Batiment,
position : Personne -> Piece,
event : Evenement
}

/*-------------------------
    INITIALISATION
-------------------------*/
pred Init [E:Etat]
{
-- aucune autorisation, sauf être à l'extérieur
E.autorisation = Personne -> Exterieur
-- toute personne est à l'extérieur du batiment
E.position = Personne -> Exterieur
E.event = init
}

/*-------------------
      Autoriser
  -------------------*/
pred Autoriser[p:Personne, b:Batiment,E,E':Etat]
{
-- précondition
b != Exterieur
not (p->b in E.autorisation) -- pas deja autorisé
-- maj etat
E'.autorisation = E.autorisation + p->b
-- nochange
E'.position = E.position
E'.event=autoriser
```

```
}

/*-------------------
      Revoquer
  -------------------*/
pred Revoquer[p:Personne, b:Batiment,E,E':Etat]
{
-- précondition
b != Exterieur
-- autorisé à ce batiment
p->b in E.autorisation
-- ne doit pas revoquer de sortir du batiment
let pi_aut = p.(E.autorisation-p->b).*~estDans |
    Exterieur in (p.(E.position)).*(porte & pi_aut->pi_aut)
-- maj etat
E'.autorisation = E.autorisation - p->b
-- nochange
E'.position = E.position
E'.event=revoquer
}

/*-------------------
      Deplacer
  -------------------*/
pred Deplacer[pe:Personne, pi:Piece,E,E':Etat]
{
-- précondition
pi != pe.(E.position)
-- autorisé à cette pièce
pe->pi in (E.autorisation).*~estDans
-- porte vers cette pièce à partir de la pièce actuelle
pe.(E.position)->pi in porte
-- maj etat
E'.position = E.position ++ pe->pi
-- nochange
E'.autorisation = E.autorisation
E'.event=deplacer
}

pred Inv[E:Etat]
{
-- chaque personne est dans une pièce
E.position.Piece = Personne
-- position est une fonction
E.position in Personne -> lone Piece
-- une personne est dans une piece autorisée
E.position in (E.autorisation).*~estDans
}
```

2. (30 pts)

(a) Spécifiez en CSP un système de gestion de ressources. Des clients veulent utiliser des ressources. Un seul client peut utiliser une ressource à la fois. Un client `c` obtient une ressource `r` en exécutant la commande `req.c.r`. Si la ressource n'est pas disponible, cette commande ne peut être exécutée. Un client libère la ressource qu'il a obtenue en exécutant la commande `lib.c.r`. Voici le type des actions

```
channel req : CLIENT.RESSOURCE
channel lib : CLIENT.RESSOURCE
```

Solution

```
AR = {| req, lib |}

Client(c) = req!c?r:RESSOURCE -> lib!c!r -> Client(c)
Ressource(r) = req?c:CLIENT!r -> lib!c!r -> Ressource(r)

MAIN =
        (||| c : CLIENT @ Client(c))
    [| AR |]
        (||| r : RESSOURCE @ Ressource(r))
```

(b) Adaptez le système de la question 3(a) en simulant une file d'attente. Un client peut demander une ressource même si elle n'est pas disponible; il est alors mis dans une file d'attente. Quand la ressource devient disponible, le système l'alloue au client en tête de la file d'attente, en exécutant la commande `all.c.r`. Le client libère la ressource en exécutant la commande `lib.c.r`. Voici le type de la nouvelle action.

```
channel all : CLIENT.RESSOURCE
```

En CSP, une file peut-être manipulée avec les opérations suivantes:

`<>` : file vide
`<a,…,c>` : file contenant les éléments `a,b,c`, où `a` est en tête.
`head(f)` : retourne la tête de la file `f`; ex: `head(<a,b,c>) = a`
`tail(f)` : retourne la file f amputée de son premier élément; ex: `tail(<a,b,c>) = <b,c>`

Solution

```
AR = {| req, all, lib |}

Client(c) = req!c?r:RESSOURCE -> all!c!r -> lib!c!r -> Client(c)

Ressource(r,f) =    #(f) < card(CLIENT) & req?c:CLIENT!r -> Ressource(r,f^<c>)
                []
                    f != <> & all!head(f)!r -> Ressource(r,f)
                []
                    f != <> & lib!head(f)!r -> Ressource(r,tail(f))

MAIN =
        (||| c : CLIENT @ Client(c))
    [| AR |]
        (||| r : RESSOURCE @ Ressource(r,<>))
```

3. (20 pts) Considérez l'expression de processus `P` ci-dessous:

```
P = ((a->SKIP [] c->SKIP) |[{a}]| (a->SKIP [] b->SKIP));d->SKIP
```

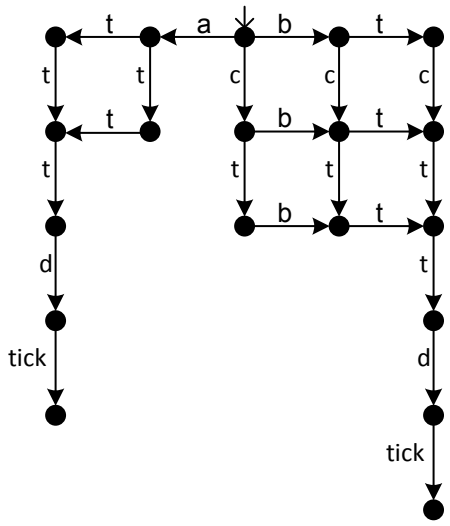
(a) Prouvez la transition `P -a-> P'`

Solution

```
_______________ ->                          _______________ ->
a->SKIP -a-> SKIP                           a->SKIP -a-> SKIP
_____________________[]                     _____________________[]
a->SKIP [] c->SKIP -a-> SKIP                a->SKIP [] b->SKIP -a-> SKIP
_____________________________________________________________________|[]|
(a->SKIP [] c->SKIP) |[{a}]| (a->SKIP [] b->SKIP) -a->  SKIP |[{a}]| SKIP
_____________________________________________________________________;
((a->SKIP [] c->SKIP) |[{a}]| (a->SKIP [] b->SKIP));d->SKIP -a-> (SKIP |[{a}]| SKIP);d->SKIP
```

(b) Donnez le graphe de transition de `P`.

Solution



4. (20 pts)

Prouvez les formules suivantes en utilisant les règles d'inférence de la logique propositionnelle.

(a)

$$((A \vee B) \Rightarrow C) \Rightarrow ((A \Rightarrow C) \wedge (B \Rightarrow C))$$

Solution

$$\dfrac{\dfrac{\dfrac{\dfrac{\lceil (A \vee B) \Rightarrow C \rceil^{[1]} \quad \dfrac{\lceil A \rceil^{[2]}}{A \vee B}\,[\mathrm{I}_{\vee 2}]}{C}\,[\mathrm{E}_{\Rightarrow}]}{A \Rightarrow C}\,[\mathrm{I}_{\Rightarrow}]^{[2]} \quad \dfrac{\dfrac{\lceil (A \vee B) \Rightarrow C \rceil^{[1]} \quad \dfrac{\lceil B \rceil^{[3]}}{A \vee B}\,[\mathrm{I}_{\vee 1}]}{C}\,[\mathrm{E}_{\Rightarrow}]}{B \Rightarrow C}\,[\mathrm{I}_{\Rightarrow}]^{[3]}}{(A \Rightarrow C) \wedge (B \Rightarrow C)}\,[\mathrm{I}_{\wedge}]}{((A \vee B) \Rightarrow C) \Rightarrow ((A \Rightarrow C) \wedge (B \Rightarrow C))}\,[\mathrm{I}_{\Rightarrow}]^{[1]}$$

(b)

$$((A \Rightarrow C) \wedge (B \Rightarrow C)) \Rightarrow ((A \vee B) \Rightarrow C)$$

Solution

$$\dfrac{\dfrac{\lceil A \vee B \rceil^{[1]} \quad \dfrac{\lceil A \rceil^{[2]} \quad \dfrac{\lceil (A \Rightarrow C) \wedge (B \Rightarrow C) \rceil^{[3]}}{A \Rightarrow C}\,[\mathrm{E}_{\wedge 1}]}{C}\,[\mathrm{E}_{\Rightarrow}] \quad \dfrac{\lceil B \rceil^{[2]} \quad \dfrac{\lceil (A \Rightarrow C) \wedge (B \Rightarrow C) \rceil^{[3]}}{B \Rightarrow C}\,[\mathrm{E}_{\wedge 2}]}{C}\,[\mathrm{E}_{\Rightarrow}]}{C}\,[\mathrm{E}_{\vee}]^{[2]}}{(A \vee B) \Rightarrow C}\,[\mathrm{I}_{\Rightarrow}]^{[1]}}{((A \Rightarrow C) \wedge (B \Rightarrow C)) \Rightarrow ((A \vee B) \Rightarrow C)}\,[\mathrm{I}_{\Rightarrow}]^{[3]}$$

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