
	<p>ANNÉE : 2010/2011</p> <p>PARCOURS : Master 1 Informatique UE : INF 471, Conceptions Formelles Module : Modélisation et Vérification Date : Lundi 11 avril 2011 Heure : 14 heures Durée conseillée : 1 heure 30 Documents : autorisés Épreuve de M. Alain GRIFFAULT</p>	 <p>U. F. R. Mathématiques Informatique</p>
---	---	--

Code d'anonymat :

Avertissement

- La plupart des questions sont indépendantes.
- Le barème total est de 23 points car le sujet est assez long.
- L'espace laissé pour les réponses est suffisant (sauf si vous utilisez ces feuilles comme brouillon, ce qui est fortement déconseillé).

Conception d'un contrôleur de machine à café (23 points)

L'objectif de l'exercice est de concevoir le contrôleur d'un distributeur de café.

Le distributeur peut délivrer des cafés avec ou sans sucre. Un utilisateur doit d'abord choisir sa boisson en appuyant sur le bouton correspondant. Si le bouton s'allume, cela signifie que ce choix est possible. L'utilisateur doit alors introduire des pièces jusqu'à atteindre le prix de la boisson. Aussitôt que cette valeur est atteinte, la boisson est servie, et tous les boutons s'éteignent. L'utilisateur peut interrompre le processus grâce à une action "reset", ce qui annule son choix et retourne les pièces déjà introduites. Par contre, l'utilisateur ne peut pas modifier son choix de boisson si une pièce est déjà dans le monnayeur.

Chaque boisson servie demande des ressources : une dose de café, un verre, et éventuellement une dose de sucre ; mais aussi que les pièces introduites soient stockées dans une boîte. Les ressources peuvent venir à manquer, et la boîte peut devenir pleine. Un technicien de maintenance assure les réparations du distributeur lorsqu'un service n'est plus possible. Pendant son intervention, le distributeur n'est pas opérationnel.

L'approche suivie pour la conception du contrôleur est la suivante :

1. Modélisation et validation des composants du distributeur
 - Les ressources en café, sucre et verres.
 - La boîte qui reçoit les pièces.
 - Les boutons pour choisir entre café et café sucré.
 - Le monnayeur pour introduire les pièces.
2. Modélisation et validation des panneaux de commandes.
 - Le panneau utilisateur et son contrôleur.
 - Le panneau pour la maintenance et son contrôleur.
3. Modélisation et validation du distributeur
4. Vérification de quelques exigences.

Exercice 1 (Conception du modèle (16 points))

La description ALTARICA utilise quelques constantes :

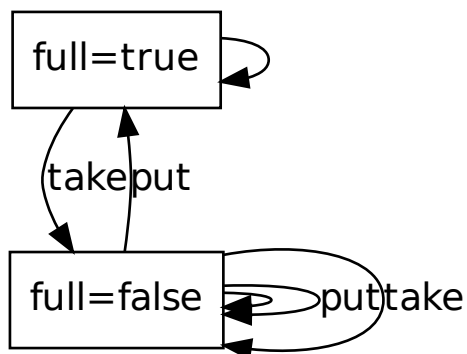
```
Const coffeePrice : integer = 2;
Const sugarPrice : integer = 1;
Const glassPrice : integer = 0;
Const maxPrice : integer = coffeePrice+sugarPrice+glassPrice;
```

La description ALTARICA d'une cartouche pour les ressources est la suivante :

```
// Cartridge modelise coffee, sugar and glass ressources
node Cartridge
  state empty : bool : public;
  init empty := false;
  event fill, get
  trans true  |- fill -> empty := false;
             ~empty |- get -> empty := false;
             ~empty |- get -> empty := true;
edon
```

Question 1.1 (1,5 points) Dessinez la sémantique du composant Cartridge.

La sémantique du composant Box est donnée par le graphe suivant :



Question 1.2 (1,5 points) Donnez le code ALTARICA correspondant.

```
// Box modelise the oject that receive coins
node Box
  state full : bool : public;
  init full := false;
  event put, take
  trans
```

edon

Question 1.3 (1,5 points) Ajoutez des commentaires et complétez les synchronisations dans la description ALTARICA du composant Stock.

```
node Stock
  sub C, S, G : Cartridge;    // C means Coffee, S Sugar and G Glass
    M : Box;                  // M means Money
  event coffee, coffeeSugar, maintenance
  trans true |- coffee, coffeeSugar, maintenance -> ;
  /* Comment on synchronisation
  *
  *
  */
  sync <coffee, C.get, G.get, M.put>;

    <coffeeSugar,                >;

    <maintenance, C.fill, S.fill, G.fill, M.take>;
edon
```

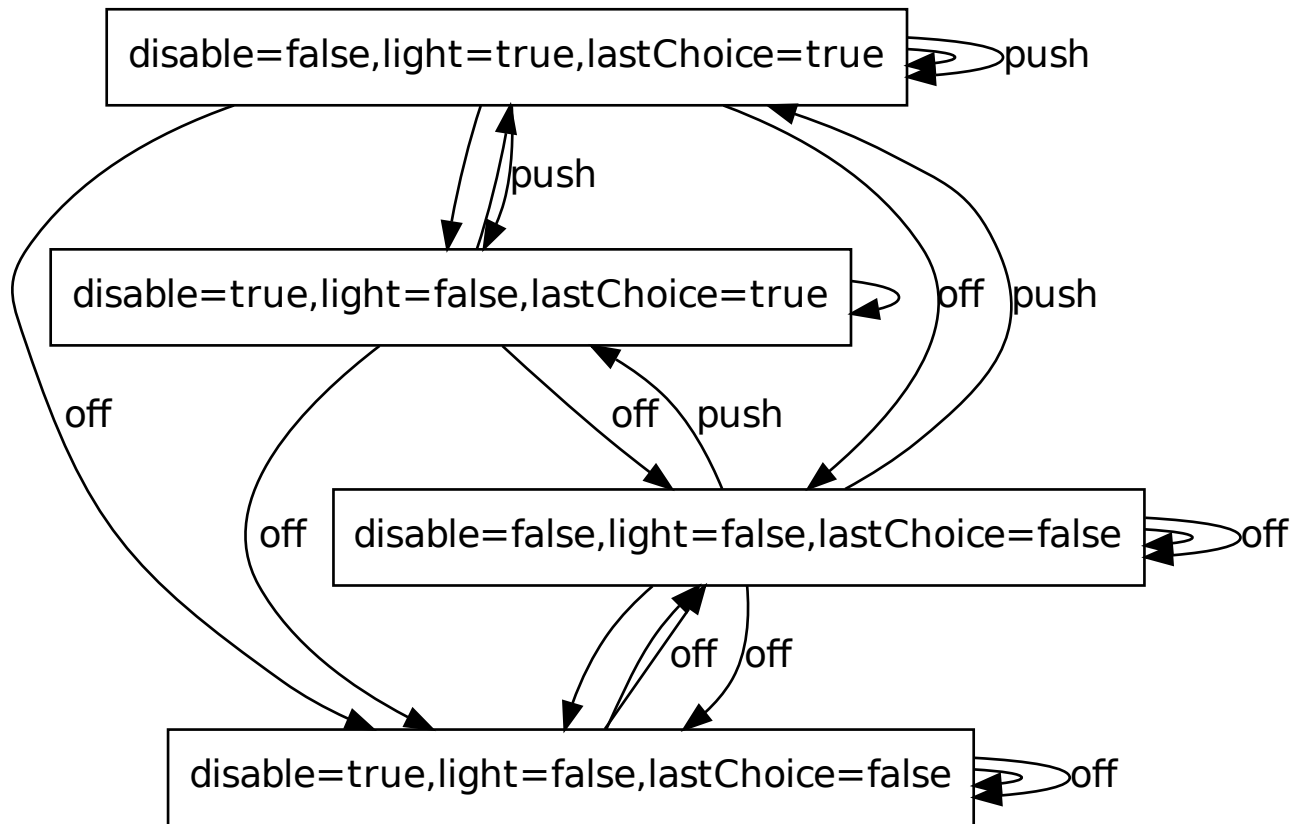
Question 1.4 (2 points) Les modèles Cartridge, Box et Stock sont tous non déterministes (l'action get peut conduire dans la situation vide ou non pour le composant Cartridge, et l'action put peut remplir ou non le composant Box). Expliquez l'avantage d'utiliser du mon déterminisme pour la modélisation des stocks.

La description ALTARICA du monnayeur est la suivante :

```
// Coin modelise the input mecanism for the coins
node Coin
  // valueChoice wil be constraint by the environment depending on the drink
  flow valueChoice : [0,maxPrice];
  state value : [0,maxPrice] : public;
  init value := 0;
  event reset, accept, deliver
  trans true |- reset -> value := 0;
    value < (valueChoice-1) |- accept -> value := value + 1;
    value = (valueChoice-1) |- deliver -> value := 0;
  // To understand the semantic, uncomment the following line
  // assert valueChoice = 3;
edon
```

Question 1.5 (2 points) Dessinez la sémantique du composant Coin, dans le cas particulier de valueChoice = 3. Pour cela vous pouvez momentanément imaginer que la ligne assert valueChoice = 3 n'est pas un commentaire.

La sémantique du composant Button est donnée par le graphe suivant :



Question 1.6 (2 points) Donnez le code ALTARICA correspondant.

```
// Button to make choice between differents beverages
node Button
  // disable will be constraint by the environment
  // disable depends on ressources and maintenance
  flow disable, light : bool : public;
  state lastChoice : bool : public;
  init lastChoice := false;
  event push, off;
  trans
```

```
  assert light = (~disable & lastChoice);
edon
```

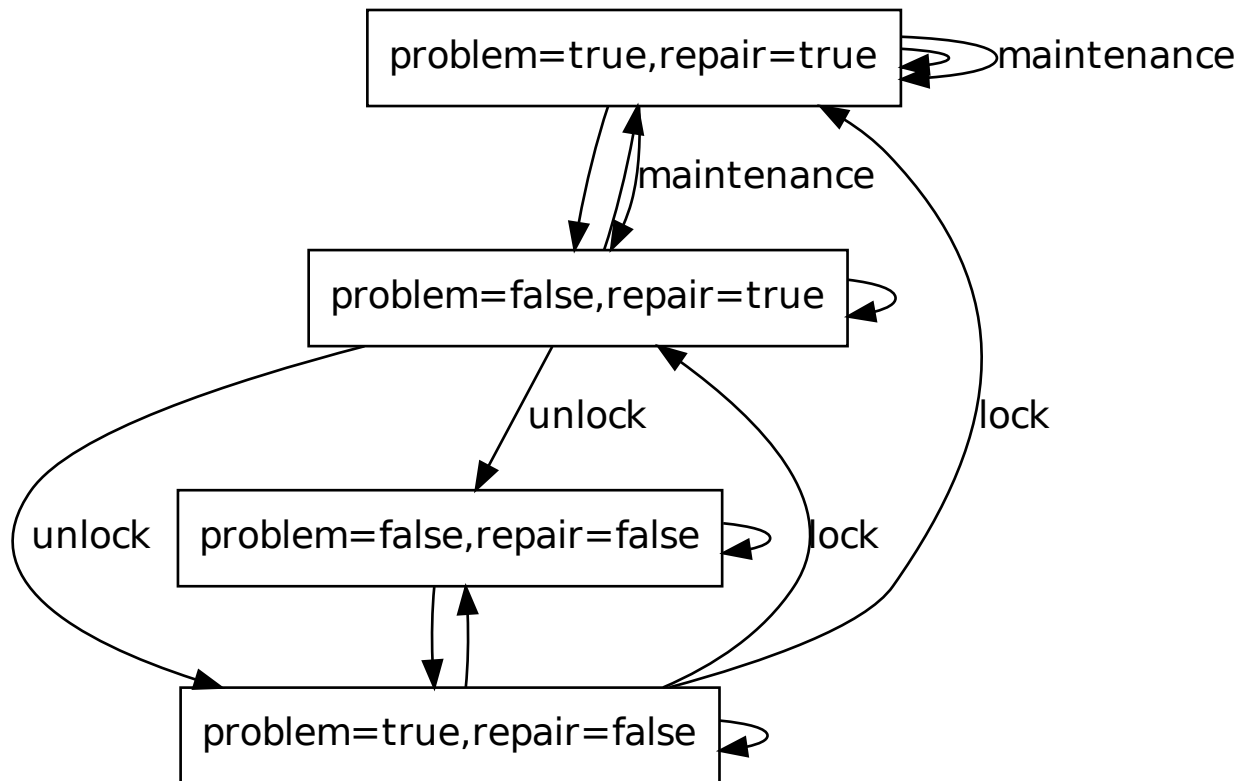
Question 1.7 (1,5 points) Ajoutez des commentaires dans la description ALTARICA du composant UserPanel.

```

node UserPanel
  // C means Coffee, CS Coffee with Sugar
  sub C, CS : Button;
    M : Coin;
  // disable will be constraint by the environment
  flow disable : bool;
  event select_C, select_CS, coin, reset : public;
    deliver_C, deliver_CS;
  /* Comments on trans
  *
  *
  *
  *
  *
  *
  *
  */
  trans ~disable & M.value = 0 |- select_C, select_CS -> ;
    ~disable & C.light      |- coin, deliver_C -> ;
    ~disable & CS.light     |- coin, deliver_CS -> ;
    ~disable & M.value > 0 |- reset -> ;
  /* Comments on synchronisations
  *
  *
  *
  *
  *
  *
  *
  */
  sync <select_C, C.push, CS.off>;
    <select_CS, C.off, CS.push>;
    <coin, M.accept>;
    <deliver_C, C.off, CS.off, M.deliver>;
    <deliver_CS, C.off, CS.off, M.deliver>;
    <reset, C.off, CS.off, M.reset>;
  /* Comments on the assertion
  *
  *
  *
  *
  *
  *
  *
  */
  assert C.light => (M.valueChoice = coffeePrice + glassPrice);
    CS.light => (M.valueChoice = coffeePrice + sugarPrice + glassPrice);
    ~(C.light | CS.light) => (M.valueChoice = 0);
edon

```

La sémantique du composant `OperatorPanel` est donnée par le graphe suivant :



Question 1.8 (2 points) Donnez le code ALTARICA correspondant.

```

// OperatorPanel to control the behaviour of the operateur
node OperatorPanel
  // problem will be constraint by the environment
  flow problem : bool;
  // repair is true during the maintenance
  state repair : bool : public;
  init repair := false;
  // lock to start the maintenance, unlock to finish it
  event lock, maintenance, unlock;
  trans

```

edon

Question 1.9 (2 points) Ajoutez des commentaires dans la description ALTARICA du composant CoffeeMachine.

```
node CoffeeMachine
  /* Comments on the hierarchy
  *
  *
  *
  *
  *
  *
  *
  */
  sub UP : UserPanel;
    OP : OperatorPanel;
    S : Stock;
  /* Comments on the synchronisations
  *
  *
  *
  *
  *
  *
  *
  */
  sync <OP.maintenance, S.maintenance>;
    <UP.deliver_C, S.coffee>;
    <UP.deliver_CS, S.coffeeSugar>;
  /* Comments on the assertion
  *
  *
  *
  *
  *
  *
  *
  */
  assert UP.C.disable = (S.C.empty | S.G.empty | S.M.full);
    UP.CS.disable = (S.C.empty | S.S.empty | S.G.empty | S.M.full);
    OP.problem = (UP.M.value = 0 & (S.C.empty | S.S.empty | S.G.empty | S.M.full));
    OP.repair = UP.disable;
edon
```

Exercise 2 (Validation et verification du model (7 points))

Question 2.1 (2 points) Complétez le calcul de “notP1” dans le but de vérifier l'exigence suivante : “Il est impossible d'obtenir une boisson pendant la maintenance”.

```
// P1 : Impossible to obtain a drink during the maintenance
with CoffeeMachine do
  notP1 :=
```

```
  test(notP1,0) > '$NODENAME.verification';
done
```

Question 2.2 (2 points) Complétez le calcul de “notP2” dans le but de vérifier l'exigence suivante : “Il est impossible de modifier son choix si le monnayeur n'est pas vide”.

```
// P2 : Impossible to make a new choice if some coins have been introduced
with CoffeeMachine do
  notP2 :=
```

```
  test(notP2,0) >> '$NODENAME.verification';
done
```

Question 2.3 (3 points) Complétez le calcul de “notP3” dans le but de vérifier l'exigence suivante : “Vous devez attendre ou bien démarrer la maintenance dès qu'il n'y a plus de verre”.

```
// P3 : You must wait or start the maintenance as soon as there is no more glass
with CoffeeMachine do
  justNoGlass :=
```

```
  notP3 :=
```

```
  test(notP3,0) >> '$NODENAME.verification';
done
```