Master 1 **Première session 2011-2012**

ACF : Analyse et Conception Formelles

2 heures - Documents autorisés

Pour les définitions de types, propriétés et fonctions, on attend de la syntaxe Isabelle/HOL. Pour chaque fonction récursive justifiez sa terminaison.

Question 1. Donnez les trois propriétés (les plus générales possibles) attendues sur les deux fonctions suivantes. On attend des lemmes Isabelle/HOL ne faisant intervenir que ces deux fonctions et le type option.

```
datatype 'a option= None | Some 'a

fun nth:: "nat ⇒ 'a list ⇒ 'a option"
where
"nth _ [] = None" |
"nth 0 (x#_)= Some x" |
"nth x (y#ys)= (nth (x - 1) ys)"

fun member::"'a ⇒ 'a list ⇒ bool"
where
"member _ [] = False" |
"member e (x # xs) = (if (x=e) then True else (member e xs))"
```

Question 2. Sur le lemme Isabelle `lemma "leng(l1 @ l2) = leng (l2 @ l1)"`, l'appel de l'outil Nitpick donne la réponse suivante :

```
Nitpick found a counterexample for card 'a = 5 and card 'b = 5:

  Free variables:
    l1 = [b1]
    l2 = [b2]
    leng = (λx. _)([] := a1, [b1] := a1, [b1,b2] := a1, [b2] := a3, [b2, b1] := a2)
```

1. Expliquez en détail chaque ligne de cette réponse.
2. Que peut-on en déduire, d'un point de vue logique, sur le lemme
3. Qu'aurait-on pu déduire sur le lemme, d'un point de vue logique, si la réponse avait été de la forme : `Nitpick found no counterexample`

Question 3. Soit la fonction `normalise` suivante :

```
function normalise:: "nat list => nat list"
where
"normalise []= []" |
"normalise (x#y)=
  (if (x = 0 \/ x = 1) then (x#(normalise y)) else
   (1 # (normalise ((x - 1)#y))))"
apply pat_completeness
apply auto
done
```

1. Donnez un exemple de ce que fait cette fonction.
2. Donnez une fonction de mesure permettant de démontrer la terminaison de cette fonction.

Question 4. Expliquez ce qu'est la base de confiance (trusted base) d'une théorie Isabelle/HOL. Expliquez comment augmenter la confiance dans cette base. Dans le TP5 portant sur les sous-séquences, quelle pourrait être la base de confiance ? Pourrait-on la réduire ? Si oui, de quelle façon ?

Question 5. On souhaite représenter les tableaux en Isabelle/HOL à l'aide de listes.

1. Choisissez une représentation pour les tableaux, définissez le type `'a array` ainsi que les fonctions :
   - `get` qui, étant donné un tableau `t` de type `'a array` et un naturel `i`, rend le $\texttt{i}^{eme}$ élément de `t`.
   - `set` qui, étant donné un tableau `t` de type `'a array`, un naturel `i` et une valeur `e` de type `'a`, rend le tableau `t` dans lequel la valeur à l'indice `i` est `e`.
2. Sous la forme de lemmes Isabelle/HOL, donnez les deux propriétés (les plus générales possibles) attendues sur `set` et `get`.
3. Définissez une fonction `concat` permettant de concaténer deux tableaux de type `'a array`.
4. Sous la forme de lemmes Isabelle/HOL, donnez les deux propriétés (les plus générales possibles) attendues sur `get` et `concat`.

Question 6. Donnez trois propriétés du langage Scala qui justifient la pertinence du choix de Scala comme langage cible pour exporter les théories Isabelle/HOL. Expliquez avec vos mots chacune de ces propriétés.

Master 1 **Seconde session 2011-2012**

ACF : Analyse et Conception Formelles

2 heures - Documents autorisés

Pour les définitions de types, propriétés et fonctions, on attend de la syntaxe Isabelle/HOL. Pour chaque fonction récursive justifiez sa terminaison.

Question 1. Définir la théorie Isabelle comprenant les fonctions suivantes :

- `suppress` qui supprime toutes les occurrences d'un élément dans une liste
- `sorted` qui détermine si une liste de naturels est triée
- `fusion` qui fusionne deux listes triées de naturels en une seule liste triée.

Pour chaque fonction, justifiez sa terminaison. Donnez la propriété la plus générale que l'on peut exprimer sur chaque couple de fonction suivantes :

- `suppress` et `sorted`
- `sorted` et `fusion`

Question 2. Soit la formule ϕ correspondant au lemme Isabelle suivant :

```
"leng(l1 @ l2) = leng (l2 @ l1)"
```

Sur cette formule ϕ, l'appel de l'outil Nitpick donne la réponse suivante :

```
Nitpick found a counterexample for card 'a = 5 and card 'b = 5:

  Free variables:
    l1 = [b1]
    l2 = [b2]
    leng = (λx. _)([]:= a1, [b1]:= a1, [b1,b2]:= a1, [b2]:= a3, [b2, b1]:= a2)
```

1. Expliquez en détail chaque ligne de cette réponse.
2. D'un point de vue logique, que peut-on en déduire sur ϕ ? sur $\neg\phi$?
3. Donnez une interprétation de $\neg\phi$.

Question 3. Les files FIFO (First In First Out) sont une structure sur laquelle on définit généralement trois opérations : `add` qui ajoute un élément en queue de file, `head` qui donne l'élément en tête de file et `pop` qui supprime l'élément en tête de file. Il existe une représentation fonctionnelle des files FIFO dont l'utilisation a un coût constant en moyenne. Le principe est le suivant : une file `f` est représentée par un couple de listes `f=(l1,l2)`.

- La fonction `add e f`, d'ajout en queue de file, ajoute `e` en tête de `l1`, la première liste.

- La fonction de consultation de la tête de file `head f` n'est définie que si `f` est non vide. Sur une file non vide, la fonction `head f` rend deux résultats : l'élément en tête de file ainsi que la file elle même. Si la deuxième liste `l2` n'est pas vide, `head` rend l'élément en tête de `l2` et la file `(l1,l2)`. Si `l2` est vide, `head` rend le premier élément de `reverse l1` et la file sous la forme `([], reverse l1)`, où `reverse` est la fonction inversant une liste.
- Le principe de la fonction `pop f` qui rend la file `f` privée de son élément de tête est similaire. Cette fonction n'est définie que pour les files `f` non vides. Si dans la file `f=(l1,l2)`, la liste `l2` n'est pas vide alors elle rend `(l1, tl l2)` sinon, elle rend la file `([], tl (reverse l1))`, ou `tl` est la fonction retournant une liste privée de son premier élément.

Voici un exemple d'exécution de ces fonctions :

1. soit `f` une file vide, représentée par `([], [])` ;
2. soit `f1 = add 1 f = ([1], [])` ;
3. soit `f2 = add 3 (add 2 f1) = ([3,2,1],[])` ;
4. le résultat de `head f2` est l'élément `1` ainsi que la file `f3= ([], [1,2,3])` ;
5. si on ajoute un élément à `f3`, on obtient `f4 = add 4 f3` où `f4= ([4], [1,2,3])`
6. enfin, `pop f4= ([4], [2,3])`.

Travail à réaliser :

- Proposez une théorie Isabelle/HOL définissant le type des files, la file vide `fvide` ainsi que les fonctions `add`, `head` et `pop`.
- Donnez les 3 propriétés les plus générales attendues sur `fvide`, `add`, `head` et `pop` sous la forme de lemmes en Isabelle/HOL.

Master 1 **Première session 2012-2013**

ACF : Analyse et Conception Formelles

2 heures - Documents autorisés

Pour les définitions de types, propriétés et fonctions, on attend de la syntaxe Isabelle/HOL. Pour chaque fonction récursive justifiez sa terminaison. Le barême est donné à titre indicatif.

Question 1 (3 points)**.** Pendant ce cours, vous avez développé vos applications sous forme de théories Isabelle/HOL et généré le code Scala à partir de celles-ci. Donnez trois avantages *essentiels* de cette approche par rapport à un développement effectué directement en Scala.

Question 2 (2 points)**.** Quelles propriétés doit satisfaire une variable `x` pour pouvoir faire une preuve par induction sur `x` ? Sur quel type de formule une preuve par induction est-elle nécessaire ?

Question 3 (5 points)**.** Soit la formule ϕ_1 correspondant au lemme Isabelle suivant :

```
"∀ l1. f(l1)= List.length(l1)"
```

Sur cette formule ϕ_1, l'appel de l'outil Nitpick donne la réponse suivante :

```
Nitpick found a counterexample for card 'a = 2:
  Free variables:
    f = (λx. _)([]:= 0, [a1]:= 0)
  Skolem constant:
    l1 = [a1]
```

Soit la formule ϕ_2 correspondant au lemme Isabelle suivant :

```
"leng(l1 @ l2) = leng(l2 @ l1)"
```

Sur cette formule ϕ_2, l'appel de l'outil Nitpick donne la réponse suivante :

```
Nitpick found a counterexample for card 'a = 5 and card 'b = 5:
  Free variables:
    l1 = [b1]
    l2 = [b2]
    leng = (λx. _)([]:= a1, [b1]:= a1, [b1,b2]:= a1, [b2]:= a3, [b2, b1]:= a2)
```

Pour ϕ_1 et ϕ_2 :

1. Expliquez en détail chaque ligne de la réponse de Niptick pour ϕ_1 et ϕ_2.
2. D'un point de vue logique, que peut-on en déduire sur ϕ_1, ϕ_2 ? Sont elles contradictoires, valides, satisfaisables ? Pour montrer la satisfaisabilité donnez une interprétation (domaine D, interprétation des fonctions et prédicats et valuation des variables) satisfaisant la formule.

3. Même question pour $\neg\phi_1$ et $\neg\phi_2$.

Question 4 (10 points)**.** On souhaite modéliser le fonctionnement d'une imprimante recevant des demandes et des annulations d'impression. L'imprimante reçoit des messages de la forme : `(Print i s)` ou `(Cancel i)` où `i` est l'entier naturel identifiant (de façon unique) la demande d'impression et `s` est une chaîne représentant le document à imprimer. Pour simplifier, on suppose que l'imprimante ne recevra jamais de requête d'annulation d'une impression `i` avant la demande d'impression `i`. L'imprimante gère une file FIFO (First In First Out) de documents en attente d'impression et une file FIFO d'identifiants de documents imprimés. On modélisera l'impression d'un document, identifié par `i`, par la suppression du document de la file des documents à traiter et l'ajout de `i` dans la file des documents imprimés. Le fonctionnement de l'imprimante sera régit par les règles suivantes :

(a) L'imprimante traite toujours la réception des messages `Print`/`Cancel` avant de traiter les impressions des documents en attente ;

(b) S'il n'y a pas de messages à traiter et qu'un document est en tête de la file des documents à imprimer, celui-ci est imprimé (son identifiant est ajouté dans la file des documents imprimés) ;

(c) Toute demande d'impression non annulée est réalisée ;

(d) Si l'impression d'un document identifié par `i` est demandée *puis* annulée avant que le document soit imprimé, le document n'est pas imprimé ;

(e) Les documents sont imprimés dans l'ordre de réception des messages `Print` par l'imprimante.

Dans la syntaxe d'Isabelle/HOL, définissez les types, fonctions et lemmes suivants :

1. Définissez les types nécessaires pour représenter les messages et l'état de l'imprimante.
2. Définissez la fonction qui pour un état d'imprimante et une liste de messages donne le nouvel état de l'imprimante. Pour simplifier, on suppose que la liste utilisée lors de l'appel à cette fonction contient tous les messages envoyés à l'imprimante.
3. Définissez les lemmes correspondant au règles (b), (c), (d) et (e).

Master 1 **Deuxième session 2012-2013**

ACF : Analyse et Conception Formelles

2 heures - Documents autorisés

Pour les définitions de types, propriétés et fonctions, on attend de la syntaxe Isabelle/HOL. Pour chaque fonction récursive justifiez sa terminaison.

Question 1. Définir la théorie Isabelle comprenant les fonctions suivantes :
- `suppress` qui supprime toutes les occurrences d'un élément dans une liste
- `sorted` qui détermine si une liste de naturels est triée
- `fusion` qui fusionne deux listes triées en une liste triée

Pour chaque fonction, justifiez sa terminaison. Donner les propriétés les plus générales que l'on peut exprimer sur chaque couple de fonction suivantes :
- `suppress` et `sorted`
- `sorted` et `fusion`

Question 2. Soit la fonction `memb` et le lemme `memLem` définis de la façon suivante :

```
fun memb:: "'a => 'a list => bool"
where
"memb x [] = False" |
"memb x (y#ys) = (or (x=y) (memb x ys))"

lemma memLem: "memb x [y,z,x]"
```

Donnez une définition de la fonction `or` telle que le lemme `memLem` puisse être démontré automatiquement par réécriture. Donnez la séquence de réécritures permettant de le démontrer.

Question 3. Soit la classe des programmes uniquement constitués d'affectations assignant à une variable soit un entier soit la valeur d'une autre variable. Par exemple :

```
x:= 2
y:= 1
y:= x
x:= 3
x:= z
```

On suppose que la valeur associée à une variable non définie par une affectation est -1. Par exemple, sur le programme précédent, à la fin de l'exécution la valeur de `x` est -1 et la valeur de `y` est 2 (et `z` n'est pas définie). On peut définir ce type de programmes en Isabelle/HOL de la façon suivante :

```
datatype exp= Var string | Const int
type_synonym programme= "(string * exp) list"
```

1. Définir une fonction `simplifier:: programme => programme` qui supprime dans un programme toute affectation sur une variable `v` si elle est immédiatement suivie par une affectation sur la même variable `v`. Par exemple le résultat de

   cette fonction sur le programme

   ```
   x:= 2
   y:= 1          x:= 2
   y:= x   sera   y:= x
   x:= 3          x:= z
   x:= z
   ```

2. Donnez le théorème de correction de la fonction `simplifier` : définissez les fonctions nécessaires et les lemmes Isabelle/HOL assurant qu'à la fin de l'exécution d'un programme ou de sa version simplifiée les variables définies ont les mêmes valeurs.

Master 1 **Première session 2013-2014**

ACF : Analyse et Conception Formelles

2 heures - Documents autorisés

Pour les définitions de types, propriétés et fonctions, on attend de la syntaxe Isabelle/HOL. Pour chaque fonction récursive justifiez sa terminaison. Le barême est donné à titre indicatif.

Question 1 (3 points)**.** Soit l'ensemble de prédicats $\mathcal{P} = \{R\}$ tel que R prend 2 arguments. Le but Isabelle/HOL suivant est-il prouvable ? Expliquez pourquoi.

$$[\![(\forall\, x\ y\ z.\ (R(x,y) \wedge R(y,z)) \longrightarrow R(x,z))\,;\ (\forall\, x.\neg R(x,x))\,;\ R(a,b)\,;\ R(b,a)\,]\!] \longrightarrow \text{False}$$

Question 2 (10 points)**.** On se propose de modéliser le fonctionnement d'une interface graphique 2D comprenant des fenêtres et des évènements de la souris. On considère que les fenêtres sont définies par la position (abscisse et ordonnée) de leur coin supérieur gauche et par la position de leur coin inférieur droit. Comme dans les interfaces graphiques traditionnelles, les fenêtres peuvent être empilées les unes sur les autres et se recouvrir partiellement ou totalement en fonction de leur disposition. On appellera GUI la pile de fenêtres ouvertes dans l'interface. Les évènements souris sont définis par une position et un mode. Deux modes existent : *sélection* et *fermeture*. L'effet d'un évènement souris à la position (x, y) affectera la première fenêtre de la GUI (en partant du haut de la pile) contenant le point (x, y) et seulement celle-ci. On dira que cette fenêtre capture l'évènement. L'effet d'un évènement souris *sélection* sera de faire remonter la fenêtre touchée vers le sommet de la GUI. L'effet d'un évènement souris *fermeture* sera de supprimer la fenêtre touchée de la GUI.

1. Définissez les types nécessaires à la modélisation du problème.
2. Définissez une fonction `update` qui à partir d'une GUI et d'un évènement souris donne le nouvel état de la GUI (L'utilisation de fonctions intermédiaires est possible).
3. Donnez d'abord en français puis à l'aide de formules des propriétés attendues sur `update` dans les scénarios suivants. L'utilisation de fonctions annexes est possible mais on cherchera à limiter, autant que possible, la base de confiance en réutilisant des fonctions prédéfinies en Isabelle/HOL.
   (a) Que doit-on observer si un évènement souris n'est capturé par aucune fenêtre d'une GUI ?
   (b) Que doit-on observer si un évènement *fermeture* est capturé par une fenêtre de la GUI ?
   (c) Que doit-on observer si un évènement *sélection* est capturé par une fenêtre de la GUI ?

Question 3 (4 points)**.** Définir la théorie Isabelle comprenant les fonctions suivantes :

- `suppress` qui supprime toutes les occurrences d'un élément dans une liste
- `sorted` qui détermine si une liste de naturels est triée
- `fusion` qui fusionne deux listes triées en une liste triée

Pour chaque fonction, justifiez sa terminaison. Donner les propriétés les plus générales que l'on peut exprimer sur chaque couple de fonction suivantes :

- `suppress` et `sorted`
- `sorted` et `fusion`

Question 4 (4 points)**.** Soit $\mathcal{F} = \{a, s\}$ un ensemble de symboles tel que s prend 1 argument et a en prend 0 (a est une constante). Soit l'ensemble de prédicats $\mathcal{P} = \{P\}$ tel que P prend 1 argument.

1. Soit ϕ la formule $(\exists x.\ P(x)) \rightarrow (\forall x.\ P(x))$. Expliquez pourquoi toute interprétation I de ϕ dont le domaine D a un seul élément est un modèle de ϕ.
2. Soit ϕ la formule $(a = s(s(a)) \wedge P(a) \wedge \neg P(s(a)) \wedge (\exists x. P(x) = P(s(s(s(x))))))$. Cette formule est elle valide ? Dans le cas contraire, donnez un contre-exemple

Master 1 **Seconde session 2013-2014**

ACF : Analyse et Conception Formelles

2 heures - Documents autorisés

Pour les définitions de types, propriétés et fonctions, on attend de la syntaxe Isabelle/HOL. Pour chaque fonction récursive justifiez sa terminaison. Le barême est donné à titre indicatif.

Question 1 (4 points)**.** Donnez le principe d'induction pour le type abstrait suivant :

```
datatype 'a prog = Skip | Aff string 'a | Block "'a prog" "'a prog"
```

Question 2 (4 points)**.** Soit $\mathcal{F} = \{a, s\}$ un ensemble de symboles tel que s prend 1 argument et a en prend 0 (a est une constante). Soit l'ensemble de prédicats $\mathcal{P} = \{P\}$ tel que P prend 1 argument.

1. Soit ϕ la formule $(\exists x.\ P(x)) \longrightarrow (\forall x.\ P(x))$. Expliquez pourquoi toute interprétation I de ϕ dont le domaine D a un seul élément est un modèle de ϕ.
2. Soit ϕ la formule $(a = s(s(a)) \wedge P(a) \wedge \neg P(s(a)) \wedge (\exists x. P(x) = P(s(s(s(x))))))$. Cette formule est elle valide ? Dans le cas contraire, donnez un contre-exemple.

Question 3 (12 points)**.** Calculs en représentation binaire.

1. Définissez le type `bin` représentant les nombres naturels en binaire en Isabelle/HOL ;
2. Définissez la fonction de conversion d'un naturel de type `bin` vers un naturel de type `nat` d'Isabelle/HOL ;
3. Définissez la fonction de conversion inverse d'un naturel de type `nat` vers un naturel de type `bin` ;
4. Définissez le ou les lemmes assurant la correction de vos fonctions de conversion ;
5. Définissez la fonction d'addition sur les naturels de type `bin` (sans vous servir de l'addition sur le type `nat`) ;
6. Définissez le ou les lemmes assurant la correction de votre fonction d'addition ;
7. En supposant tous vos lemmes démontrés, quelle pourrait être la base de confiance de votre théorie. Expliquez pourquoi.

Master 1 **Première session 2014-2015**

ACF : Analyse et Conception Formelles

2 heures - Documents autorisés

Pour les définitions de types, propriétés et fonctions, on attend de la syntaxe Isabelle/HOL. Pour chaque fonction récursive justifiez sa terminaison. Le barème est donné à titre indicatif.

Question 1 (13 points)**.** On souhaite formaliser le fonctionnement d'un réseau social. On se focalisera sur les objets suivants : les utilisateurs, les murs et les messages. On souhaite aussi représenter les relations d'amitié entre utilisateurs. Pour simplifier on considèrera que chaque utilisateur est ami avec lui-même. Chaque utilisateur a un mur unique sur lequel figurent tous les messages qu'il a reçus. Un message possède un auteur, un contenu et un destinataire (le possesseur du mur sur lequel il est déposé). Le fonctionnement du réseau est régi par les règles suivantes :

(a) Deux utilisateurs x et y sont amis si et seulement si x a demandé à y d'être son ami et y a demandé à x d'être son ami ;

(b) Tout utilisateur peut lire tous les messages de son mur ;

(c) Tout utilisateur peut poster sur son mur : il est l'auteur et le destinataire du message. S'il lit son mur, il y trouvera au moins les messages qu'il s'est postés ;

(d) Tout utilisateur peut poster sur le mur d'un autre utilisateur si et seulement si il est ami avec ce dernier ;

(e) Tout utilisateur x peut lire le mur d'un ami y. Mais, il ne voit que les messages envoyés à y par les amis de x.

1. Définissez tous les types nécessaires à la représentation de ce réseau social, en particulier : les types `utilisateur`, `message`, `reseauSocial`, etc. ;
2. Définissez les fonctions nécessaires à la gestion (simplifiée) de ce réseau social :
   - Une fonction `demandeAmi`:: `utilisateur` ⇒ `utilisateur` ⇒ `reseauSocial` ⇒ `reseauSocial` permettant à un utilisateur de demander à être ami avec un autre sur un réseau social ;
   - Une fonction `sontAmi`:: `utilisateur` ⇒ `utilisateur` ⇒ `reseauSocial` ⇒ `bool` permettant de savoir si deux utilisateurs sont amis ;
   - Une fonction `poster` :: `message` ⇒ `reseauSocial` ⇒ `reseauSocial` permettant à un utilisateur de poster un message sur le mur d'un autre utilisateur ;
   - Une fonction `lire` :: `utilisateur` ⇒ `utilisateur` ⇒ `reseauSocial` ⇒ `message list` permettant à un utilisateur de lire le mur d'un autre utilisateur.

3. Formalisez les règles de (a) à (e), définies ci-dessus, par des lemmes Isabelle/HOL. Comme dans le TP89, pour formaliser ces règles il est nécessaire de pouvoir parler de l'état du système (ici le réseau social) après l'exécution d'un certain nombre d'opérations. Une opération est soit l'envoi d'un message soit une demande pour être ami. Les opérations peuvent être formalisées par le type algébrique suivant :

   ```
   datatype operation=  DemandeAmi utilisateur utilisateur |
                        Poster utilisateur utilisateur string
   ```

   et par une fonction

   `execOperation`:: `operation list` ⇒ `reseauSocial` ⇒ `reseauSocial`

   qui exécute une liste d'opérations sur un réseau social. Cette fonction est à définir et appellera les fonctions définies dans la partie précédente.

Question 2 (5 points)**.** Soit la fonction `memb` et le lemme `membLem` définis de la façon suivante :

```
fun memb:: "'a => 'a list => bool"
where
"memb x [] = False" |
"memb x (y#ys) = (myor (x=y) (memb x ys))"

lemma membLem: "memb x [y,z,x]"
```

Donnez une définition de la fonction `myor` telle que le lemme `membLem` puisse être démontré automatiquement par réécriture. Donnez la séquence de réécritures permettant de le démontrer.

Question 3 (2 points)**.** Soit ϕ la formule suivante :

$$\phi = \forall (x :: nat)\ y\ z.\ ((x + y) \geq z) \longrightarrow (x * (z + 1)) + y > z$$

Sur cette formule, `nitpick` trouve un contre-exemple. Que peut-on dire de ϕ ? Peut-on dire de ϕ qu'elle est valide, satisfaisable ou contradictoire ? Même question pour $\neg\phi$?

Master 1 **Seconde session 2014-2015**

ACF : Analyse et Conception Formelles

2 heures - Documents autorisés

Pour les définitions de types, propriétés et fonctions, on attend de la syntaxe Isabelle/HOL. Pour chaque fonction récursive justifiez sa terminaison. Le barème est donné à titre indicatif.

Question 1 (3 points)**.** Pendant ce cours, vous avez développé vos applications sous forme de théories Isabelle/HOL et généré le code Scala à partir de celles-ci. Donnez trois avantages *essentiels* de cette approche par rapport à un développement effectué directement en Scala.

Question 2 (7 points)**.** On souhaite programmer et vérifier une bibliothèque de tables (map) associant des clés de type `'a` à des valeurs de type `'b`. On définit le type des tables en Isabelle/HOL de la façon suivante :

```
type_synonym ('a,'b) map= "('a * 'b) list"
```

1. Définissez en Isabelle/HOL les fonctions suivantes :
   (a) la consultation de la valeur associée à une clé dans une table : `get`
   (b) la suppression d'une clé dans une table : `remove`
   (c) la modification de la valeur associée à une clé dans une table : `modify`. Si la clé n'apparaît pas dans la table, la clé et la valeur sont ajoutées dans la table.

2. Formalisez sous la forme d'un lemme Isabelle/HOL les propriétés suivantes :
   (a) La fonction `remove` appliquée à la clé `k` et à la table `t` rend une table dans laquelle la clé `k` n'apparaît plus
   (b) La fonction `remove` appliquée à la clé `k` et à la table `t` ne supprime aucun couple (clé,valeur) dont la clé est différente de `k`
   (c) La fonction `modify` appliquée à la clé `k`, à la valeur `v` et à la table `t` rend une table dans laquelle la valeur associée à la clé `k` est `v`
   (d) La fonction `modify` appliquée à la clé `k`, à la valeur `v` et à la table `t` ne modifie aucun couple (clé,valeur) dont la clé est différente de `k`.

Question 3 (10 points)**.** Soit la classe des programmes uniquement constitués d'affectations assignant à une variable soit un entier soit la valeur d'une autre variable. Par exemple :

```
x:= 2
y:= 1
y:= x
x:= 3
x:= z
```

On suppose que la valeur associée à une variable non définie par une affectation est -1. Par exemple, sur le programme précédent, à la fin de l'exécution la valeur de `x` est -1 et la valeur de `y` est 2 (et `z` n'est pas définie). On peut définir ce type de programmes en Isabelle/HOL de la façon suivante :

```
datatype exp= Var string | Const int
type_synonym programme= "(string * exp) list"
```

1. Définir une fonction `simplifier:: programme => programme` qui supprime dans un programme toute affectation sur une variable `v` si elle est immédiatement suivie par une affectation sur la même variable `v`. Par exemple le résultat de cette fonction sur le programme

   ```
   x:= 2
   y:= 1
   y:= x
   x:= 3
   x:= z
   ```

   sera

   ```
   x:= 2
   y:= x
   x:= z
   ```

2. Donnez le théorème de correction de la fonction `simplifier` : définissez les fonctions nécessaires et les lemmes Isabelle/HOL assurant qu'à la fin de l'exécution d'un programme ou de sa version simplifiée les variables définies ont les mêmes valeurs.

Master 1 **Première session 2015-2016**

ACF : Analyse et Conception Formelles

2 heures - Documents autorisés

Pour les définitions de types, propriétés et fonctions, on attend de la syntaxe Isabelle/HOL. Pour chaque fonction récursive justifiez sa terminaison. Le barème est donné à titre indicatif.

Question 1 (12 points)**.** On souhaite formaliser le fonctionnement d'un ascenseur. On représentera l'état de l'ascenseur par un couple de valeurs : le numéro d'étage auquel se trouve la cabine et l'état des portes (ouvert/fermé). Pour simplifier, on considère que l'immeuble dans lequel l'ascenseur est utilisé ne comporte que 5 étages et un rez-de-chaussée : le rez-de-chaussée correspond à l'étage 0, le premier étage à l'étage 1, etc. Quand une personne appelle l'ascenseur à un étage donné, la cabine reçoit un programme : une liste d'ordres à exécuter. Il existe 4 ordres différents : monter d'un étage (M), descendre d'un étage (D), ouvrir les portes (O) et fermer les portes (F).

Il est possible de traiter les points 2 *et* 3 *du sujet de façon indépendante.*

1. Définissez tous les types nécessaires à la représentation de l'ascenseur, en particulier les types `etat`, `ordre`,...
2. On formalise la fonction `executer` qui exécute les listes d'ordre. Les propriétés attendues sur cette fonction sont les suivantes :
   (I) L'ascenseur ne se déplace jamais les portes ouvertes ;
   (II) L'ascenseur ne descend jamais plus bas que le rez-de-chaussée (étage 0) ;
   (III) L'ascenseur ne monte jamais plus haut que le 5ème étage.

   En Isabelle/HOL, donnez les définitions suivantes :
   (a) la fonction `executer`:: `etat` ⇒ `ordre list` ⇒ `etat` qui à partir d'un état de l'ascenseur et d'une liste d'ordres permet d'obtenir l'état de l'ascenseur après avoir exécuté la liste d'ordres.
   (b) les lemmes nécessaires pour formaliser les propriétés (I), (II) et (III) sur la fonction `executer`.
3. On souhaite définir et analyser la fonction générant une liste d'ordres pour atteindre un étage particulier.
   (a) Définissez la fonction `generer`:: `nat` ⇒ `nat` ⇒ `ordre list` qui à partir de deux entiers naturels i et j produit une liste d'ordres à exécuter pour que l'ascenseur, initialement à l'étage i rejoigne l'étage j et ouvre ses portes. Il n'est pas demandé de produire une liste minimale. Si un des entiers, i ou j, est un numéro d'étage invalide (supérieur à 5 ou inférieur à 0), cette fonction retourne la liste vide.
   (b) Démontrez la terminaison de la fonction `generer` en donnant une fonction de mesure.

(c) Selon vous, comment pourrait-on définir la propriété de correction de la fonction `generer`. Formalisez cette propriété sous la forme d'un lemme Isabelle/HOL.

Question 2 (4 points)**.** On souhaite programmer la fonction `setminus:: 'a list => 'a list => 'a list` représentant l'opération de différence ensembliste, usuellement notée '\'. L'ensemble $E \setminus F$ est l'ensemble E privé des éléments apparaissant dans F. Comme dans le cas du TP2, on représente les ensembles par des listes sans doublons. Ainsi `(setminus e f)` sera la liste `e` privée des éléments apparaissant dans la liste `f`. On suppose que l'on dispose de la fonction `member::'a => 'a list => bool` pour tester l'appartenance d'un élément à une liste (et donc à un ensemble). Un de vos camarades vous propose de formaliser la différence ensembliste par le lemme suivant :

```
lemma "(member x f) ⟶ ¬ (member x (setminus e f))"
```

1. Montrez que ce lemme n'est pas suffisant pour formaliser la différence ensembliste en donnant le code d'une fonction `setminus` incorrecte[1] `setminus` qui satisfait le lemme du dessus.
2. Proposez un ou des lemmes permettant de garantir que la seule implantation possible de `setminus` satisfera la définition de la soustraction ensembliste. Assurez-vous, en particulier, que ces lemmes ne sont pas satisfaits par la définition incorrecte de `setminus` que vous avez proposé pour la question précédente. Le code de la fonction `setminus` correcte n'est pas demandé.

Question 3 (4 points)**.** Soit la formule suivante :

$$(\exists\, x.\, P\, x) \longrightarrow (\forall\, x.\, P\, x)$$

Sur cette formule,

1. `nitpick [card=1]` ne trouve pas de contre-exemple. Expliquez pourquoi.
2. `nitpick [card=2]` trouve un contre-exemple. À votre avis quel est-il ? Donnez le domaine et l'interprétation de P.

1. le résultat de `(setminus e f)` ne correspond pas à la définition de l'ensemble $E \setminus F$, où E est l'ensemble représenté par la liste `e` et F l'ensemble représenté par la liste `f`.

Master 1 **Seconde session 2015-2016**

ACF : Analyse et Conception Formelles

2 heures - Documents autorisés

Pour les définitions de types, propriétés et fonctions, on attend de la syntaxe Isabelle/HOL. Pour chaque fonction récursive justifiez sa terminaison. Le barème est donné à titre indicatif.

Question 1 (3 points)**.** Soit la formule suivante :

$$(\exists\, x.\, P\, x) \longrightarrow (\forall\, x.\, P\, x)$$

Sur cette formule,

1. `nitpick [card=1]` ne trouve pas de contre-exemple. Expliquez pourquoi.
2. `nitpick [card=2]` trouve un contre-exemple. À votre avis quel est-il ? Donnez le domaine et l'interprétation de P.

Question 2 (7 points)**.** On souhaite programmer et vérifier une bibliothèque de tables (map) associant des clés de type `'a` à des valeurs de type `'b`. On définit le type des tables en Isabelle/HOL de la façon suivante :

```
type_synonym ('a,'b) map= "('a * 'b) list"
```

1. Définissez en Isabelle/HOL les fonctions suivantes :
   (a) la consultation de la valeur associée à une clé dans une table : `get`
   (b) la suppression d'une clé dans une table : `remove`
   (c) la modification de la valeur associée à une clé dans une table : `modify`. Si la clé n'apparaît pas dans la table, la clé et la valeur sont ajoutées dans la table.
2. Formalisez sous la forme d'un lemme Isabelle/HOL les propriétés suivantes :
   (a) La fonction `remove` appliquée à la clé `k` et à la table `t` rend une table dans laquelle la clé `k` n'apparaît plus
   (b) La fonction `remove` appliquée à la clé `k` et à la table `t` ne supprime aucun couple (clé,valeur) dont la clé est différente de `k`
   (c) La fonction `modify` appliquée à la clé `k`, à la valeur `v` et à la table `t` rend une table dans laquelle la valeur associée à la clé `k` est `v`
   (d) La fonction `modify` appliquée à la clé `k`, à la valeur `v` et à la table `t` ne modifie aucun couple (clé,valeur) dont la clé est différente de `k`.

Question 3 (10 points)**.** On souhaite modéliser le fonctionnement d'une imprimante recevant des demandes et des annulations d'impression. L'imprimante reçoit des messages de la forme : `(Print i s)` ou `(Cancel i)` où `i` est l'entier naturel identifiant (de façon unique) la demande d'impression et `s` est une chaîne représentant le document à imprimer. Pour simplifier, on suppose que l'imprimante ne recevra jamais de requête d'annulation d'une impression `i` avant la demande d'impression `i`. L'imprimante gère une file FIFO (First In First Out) de documents en attente d'impression et une file FIFO d'identifiants de documents imprimés. On modélisera l'impression d'un document, identifié par `i`, par la suppression du document de la file des documents à traiter et l'ajout de `i` dans la file des documents imprimés. Le fonctionnement de l'imprimante sera régit par les règles suivantes :

(a) L'imprimante traite toujours la réception des messages `Print`/`Cancel` avant de traiter les impressions des documents en attente ;

(b) S'il n'y a pas de messages à traiter et qu'un document est en tête de la file des documents à imprimer, celui-ci est imprimé (son identifiant est ajouté dans la file des documents imprimés) ;

(c) Toute demande d'impression non annulée est réalisée ;

(d) Si l'impression d'un document identifié par `i` est demandée *puis* annulée avant que le document soit imprimé, le document n'est pas imprimé ;

(e) Les documents sont imprimés dans l'ordre de réception des messages `Print` par l'imprimante.

Dans la syntaxe d'Isabelle/HOL, définissez les types, fonctions et lemmes suivants :

1. Définissez les types nécessaires pour représenter les messages et l'état de l'imprimante.
2. Définissez la fonction qui pour un état d'imprimante et une liste de messages donne le nouvel état de l'imprimante. Pour simplifier, on suppose que la liste utilisée lors de l'appel à cette fonction contient tous les messages envoyés à l'imprimante.
3. Définissez les lemmes correspondant au règles (b), (c), (d) et (e).

Master 1 **Première session 2016-2017**

ACF : Analyse et Conception Formelles

2 heures - Documents autorisés

Pour les définitions de types, propriétés et fonctions, on attend de la syntaxe Isabelle/HOL. Le barème est donné à titre indicatif.

Question 1 (10 points)**.** On souhaite réaliser une fonction filtrant une liste de e-mails à partir d'une requête. Pour simplifier, on représentera un e-mail uniquement par son destinataire, son expéditeur et son sujet. Une requête est une expression construite à partir de tests de base (sur le destinataire, l'expéditeur ou le sujet), d'un opérateur de négation, et d'opérateurs binaires (et, ou). Les tests de base sont paramétrés par une *valeur de recherche* : une chaîne de caractère. Les tests sont de trois genres possibles : test de destinataire, test d'expéditeur ou test de sujet. Par exemple, pour passer avec succès un test de destinataire, un e-mail doit avoir un destinataire est égal à la *valeur de recherche* du test. Enfin, dans une requête, on peut combiner tous les tests de base à l'aide des opérateurs et, ou et de négation. Par exemple, on veut pouvoir exprimer les requêtes suivantes :

— quels sont les e-mails qui dont le sujet est "examen d'ACF" ;

— quels sont les e-mails dont l'expéditeur est `steven.derrien@univ-rennes1.fr` et dont le destinataire n'est pas `cedric.tedeschi@univ-rennes1.fr` ;

— quels sont les e-mails dont le sujet est "important" et dont l'expéditeur n'est ni `steven.derrien@univ-rennes1.fr` ni `cedric.tedeschi@univ-rennes1.fr`.

— etc.

Le type de la fonction `filtrer` est :

```
filtrer::"requete ⇒ email list ⇒ (email list * email list)"
```

A partir d'une requête `r` et d'une liste de e-mails `l`, cette fonction rend un couple de listes. La première liste contient tous les e-mails de `l` qui satisfont `r`, la seconde contient le reste : tous les e-mails de `l` qui ne satisfont pas `r`.

1. Définissez les types nécessaires à la représentation des requêtes et des e-mails.
2. A l'aide de lemmes Isabelle/HOL, formalisez les propriétés suivantes. Si nécessaire, vous pouvez définir des fonctions supplémentaires pour faciliter la définition de ces propriétés. Cependant, il est conseillé de limiter la base de confiance.
   (a) La fonction `filtrer` ne perd pas de mail ;
   (b) La fonction `filtrer` n'ajoute pas de mail
   (c) La fonction `filtrer` ne duplique pas de mail ;
   (d) Pour une requête `r`, le filtrage avec `r` et avec la négation de `r` rendent des résultats symétriques ;

(e) Dans les deux listes résultant du filtrage, les e-mails apparaissent dans le même ordre que dans la liste initiale.

3. Proposez une définition de la fonction `filtrer`. Pour chaque fonction récursive introduite, donnez les arguments nécessaires pour prouver sa terminaison sous la forme d'une fonction de mesure.

Question 2 (3 points)**.** On définit la fonction `f` suivante.

```
fun f::"'a list ⇒ bool"
where
"f [] = True" |
"f (x#y#z) = f (y#z)"
```

Sur la définition de `f`, Isabelle/HOL nous signale la chose suivante :

```
Missing patterns in function definition:
⋀v. f [v] = undefined
```

Expliquez en détail ce que signifie ce message. D'autre part sur l'appel `value "f [a,b]"` on obtient le résultat `"f [b]" :: "bool"`. Expliquez comment la réécriture de `f [a,b]` s'achève sur ce résultat.

Question 3 (3 points)**.** Quel sont les principes d'induction qui seront associés aux types abstraits suivants par Isabelle/HOL :

1. `datatype expression = Const int | Add expression expression`
2. `datatype color = Red | Blue | Green`

Question 4 (4 points)**.** Sur le lemme Isabelle `lemma "g (l1 @ l2) = g (l2 @ l1)"`, l'appel de l'outil Nitpick donne la réponse suivante :

```
Nitpick found a counterexample for card 'a = 5 and card 'b = 5:

  Free variables:
    l1 = [b1]
    l2 = [b2]
    g = (λx. _)([] := a1, [b1] := a1, [b1,b2] := a1, [b2] := a3, [b2, b1] := a2)
```

1. Expliquez en détail chaque ligne de cette réponse.
2. Avec les informations données par Nitpick, que peut-on en déduire sur le lemme d'un point de vue logique ? Est-il contradictoire, satisfaisable, valide ?
3. Avec ces mêmes informations sait-on si la négation du lemme est contradictoire, satisfaisable, valide ?
4. Qu'aurait-on pu déduire sur le lemme, d'un point de vue logique, si la réponse avait été de la forme : `Nitpick found no counterexample`