TD de Test Logiciel

Introduction

Exercice 1 (Petites questions)**.**

1. *On dit souvent que le test exhaustif est impossible. Que serait ce test exhaustif ? Pourquoi serait-ce impossible ?*
2. *Un défaut du test aléatoire est qu'il peine à produire certaines DT très particulières. Par exemple, quelle est la probabilité de produire (x,y) avec x=y ? (machine 32 bits)*
3. *Dites pourquoi un logiciel qui a passé avec succès tous les tests système peut échouer sur les tests d'acceptation ?*
4. *Considérons un programme avec un petit nombre de chemins, et un jeu de tests couvrant tous les chemins. Les tests passent sans problème. Le programme peut-il contenir des erreurs ?*
5. *Que faire si une campagne de tests ne trouve que très peu de bugs dans un programme ?*

Exercice 2 (Un peu de test en vrai)**.**

1. *écrire une méthode Java (ou OCaml, ou autre) de signature*
   ```
   public static Vector unionSet (Vector a, Vector b)
   ```
   retournant un vecteur d'objets contenus dans `a` *ou* `b`
2. *Cet énoncé présente des défauts et des ambiguités. Trouvez-en autant que possible.*
3. *Créer un jeu de tests (CT/DT/oracle/script de test) pour détecter les bugs potentiellement introduits à cause de ces ambiguités / défauts de spécification. Documenter ces tests en décrivant de manière concise chacun d'entre eux. Jouer les tests contre votre implantation.*
4. *Réécrire la spécification pour lever les problèmes.*

Sélection de tests boîte noire

Exercice 3 (Approche pairwise)**.** *On veut s'assurer qu'une installation réseau fonctionne bien. Les variables sont l'OS, le réseau, l'imprimante et l'application.*

OS	*Réseau*	*Imprimante*	*Application*
XP	*IP*	*HP35*	*Word*
Linux	*wifi*	*Canon900*	*Excel*
MacX	*Bluetooth*	*Canon-EX*	*PowerPoint*

1. *Combien y a-t-il d'entrées possibles au problème ? Combien y a-t-il de paires de valeurs possibles ?*
2. *Combien faut-il de tests au minimum pour couvrir ces paires ? Proposer un nombre minimal / maximal de tests assurant que chaque paire est choisie. Commentez.*
3. *Cas général : soit m variables sur des domaines de taille n. Combien y a-t-il de combinaisons possibles ? Combien y a-t-il de paires de valeurs possibles ? Quel est le nombre maximal de paires couvertes par un test ? Commentez.*

Exercice 4 (Analyse partitionnelle)**.** *Donner des classes d'équivalences pour les domaines suivants :*
- *nombre de stylos*
- *nom de planète*
- *tableau de 10 entiers*

Donner des entrées pour le test aux limites, préciser les cas de base.

Exercice 5. *Soit la spécification suivante :*

```
public static int search (List list, Object elt)
// Effects: if list or elt is null throw NullPointerException
// else if elt is in the list then return the indice of one of its positions
// else return -1
```

On considère la partition basée sur les caractéristiques et blocks suivants :
C1 : place de elt dans list
- *elt est en tête de list*
- *elt est en fin de list*
- *elt est dans une position autre que tête/queue*

1. *Ce type de partition est-il basé sur l'interface ou sur les fonctionalités ?*
2. *Montrez que la partition de C1 n'est pas disjointe.*
3. *Montrez que la partition de C1 n'est pas complète.*
4. *Proposez une nouvelle partition disjointe et complète*
5. *Donner maintenant une partition en vous basant uniquement sur l'interface (donc les types).*

Sélection de tests boîte blanche

Exercice 6 (Critères de couverture orientés contrôle)**.** *Soit le programme ci-dessous.*

```
1
2 void foo(bool a, bool b, bool c) {
3   if (a or (b and c)) then
4       println(``ok'');
5   else
6       ();
7   println("fin");
8 }
```

(1) Pour chacun des critères ci-dessous, donner les éléments à couvrir : instructions (I), décisions (D), conditions (C), décisions/conditions (DC), conditions multiples (MC), MC/DC.

(2) Donner des jeux de tests du programme couvrant les critères et montrant qu'ils sont différents.

Exercice 7 (Couverture structurelle)**.** *Soit le programme C suivant :*

```
1  /* Outputs result = 0+1+...+|value|
2   * if result > maxint then error
3   */
4  void maxsum(int maxint, int value) {
5    int result =0;
6    int i =0;
7    if (value < 0)
8      value = -value;
9    while (i< value && result <= maxint) {
10     i++;
11     result = result+i;
12   }
13   if (result <= maxint)
14      println(result);
15   else
16      println("error");
17 }
```

1. *Donner le graphe de contrôle du programme.*
2. *Donner une suite de tests TS_n qui couvre tous les noeuds du graphe de contrôle. Justifier votre réponse.*
3. *La suite TS_n couvre-t-elle tous les arcs ? Si oui, indiquer les données de tests qui effectuent la couverture des arcs. Sinon, ajouter des données de test pour obtenir une suite de tests TS_a qui couvre tous les arcs.*
4. *Indiquer quelles sont les lignes de code correspondant aux définitions de la variable `result` (ensemble defs(`result`)). Pareil pour les ensembles d'utilisation en calcul c-use(`result`) et d'utilisation en prédicats p-use(`result`).*
5. *Donner une suite de test TS_d qui couvre le critère all-use-one-def pour les p-use de `result`.*
6. *Explicitez les chemins à couvrir pour le critère all-use-all-def pour les p-use de `result`.*

Mutations

Exercice 8. *Soit le programme C de l'exercice 7.*

1. *Générer 5 mutants (ordre 1) distinguables au moyen de ROR et ABS.*
2. *Générer les DT pour tuer ces mutants, calculer le score de mutation obtenu, calculer les couvertures I, C, D obtenues.*
3. *Prenez les DT que votre voisin a trouvé pour la question précédente, calculer le score de mutation de ces DTs sur vos mutants.*
4. *Générer 1 mutant non distinguable avec ROR et 1 mutant non distinguable avec ABS.*
5. *Reprendre chacun de vos jeux de tests de l'exo 7, et calculer son score de mutation.*

Exercice 9. *Définissez un opérateur de mutations O tel que un jeu de tests O-adéquat couvre toutes les instructions. Idem avec décisions et conditions.*

Exécution symbolique

Exercice 10. *Lancer (à la main) une exécution symbolique (algo basique) sur le programme suivant et donner les différents prédicats de chemin obtenus, ainsi que des DT possibles pour chaque chemin.*

```
input(x,y,z);
assert(z>5);
a := x*x;
b := y-z;
if{a<b}
  then  c := b-a;
  else  c := a-b;
assert(c>=0)
end
```

Exercice 11.

Questions :

1. *Ajouter une commande* `switch` *dans le programme. On pourra utiliser des commandes qui pour un switch donnent le nombre de successeurs, pour chaque numéro de successeur → un couple (valeur, noeud cible), et le noeud du cas par défaut.*
2. *Modifier l'algorithme de base pour ajouter : un time out sur le solveur, une borne sur la longueur des chemins, un critère d'arrêt (couverture des instructions).*
3. *(a) Ajouter la gestion des appels de fonction. Dans ce cas, on ajoute une instruction* `var := call f` $x_1, \ldots, x_n$ *avec* `f : int*...*int -> int` *et* `return var`*. Traitez les dans l'exécution symbolique. La commande* `f.start` *donnera l'instruction de départ de la fonction* `f`*, la commande* `f.args` *donnera la liste de paramètres formels de la fonction.*

   (b) Montrer en quoi cette gestion des fonctions (dite inline*) augmente la combinatoire des chemins de la fonction appelante.*

Exécution concolique

Exercice 12. *Expliquez les deux avantages majeures de l'exécution concolique par rapport à l'exécution symbolique.*

Exercice 13. *Nous rappelons que l'exécution concolique revient à mener en parallèle une exécution symbolique et une exécution concrète, le résultat de l'exécution concrète pouvant servir à aider l'exécution symbolique.*

- *Modifier l'algorithme concolique simple pour approximer des contraintes. Concrètement, on supose maintenant que le solveur ne peut pas gérer la multiplication entre plusieurs variables. Ajouter un mécanisme de concrétisation pour pallier cela.*
- *Modifier votre algorithme concolique pour éviter les appels de fonction. On considère des fonction sans effet de bord. Proposez une technique correcte pour "concrétiser" l'appel.*
- *Donnez deux cas d'utilisation de ce genre de technique.*
- *Quel est le problème en cas d'effets de bord ?*

Exercice 14 (Heuristiques de recherche)**.** – *proposez des fonctions de calcul de score pour simuler les heuristiques suivantes : dfs, bfs, random prefix, dfs itérée (on fixe une profondeur k et on fait une dfs jusque cette borne, puis on continue en augmentant k)*