Automatisation et Tests en Programmation

Séance 1 de cours/TD

IUT SD Aurillac

Objectifs:

- Introduction au concept de Test
- Etude des tests structurels

1 Introduction

Vérification et Test [Cours]

La validation d'un système informatique (décisionnel ou non) est une étape indispensable du développement logiciel dont le rôle est de détecter les éventuelles fautes du système par rapport à sa spécification (les objectifs et contraintes qu'il doit vérifier).
On peut diviser les méthodes de validation en deux catégories :

1. La Vérification, qui consiste en un ensemble de méthodes formelles permettant de prouver mathématiquement que le programme correspond à sa spécification.

2. Le Test, qui consiste en une évaluation du système en étudiant ses exécutions.

Il n'existe pas (et ne peut exister) de méthodes de Vérification générale qui fonctionnerait sur tout système informatique. C'est pourquoi le Test est un domaine clé à maitriser.

Le déroulement d'un test est en général le suivant :

1. On commence par générer un jeu de test, un ensemble de données d'entrée qui seront fourni au programme à tester
2. On se place dans le scénario de test, si on veut pouvoir tester le programme dans des conditions particulières (mémoire faible,...)
3. On exécute ce scénario pour observer un résultat.
4. On compare (de manière automatique ou manuelle) ce résultat au résultat attendu.

Classification des Tests [Cours]

Il existe une grande variété de forme de test, selon le niveau de détail ou les outils appliqués. Voici quelques paramètres utilisés pour classifier les tests :

1. Le premier paramètre de classification est le Niveau de Test, c'est à dire à quel niveau de détail dans l'application on va effectuer les tests
   - Test unitaire : On teste chaque composant logiciel de manière individuelle (par exemple, chaque fonction ou procédure).

   - Test d'intégration : On s'assure que les interfaces (les signatures) des divers composants sont cohérentes et que leur combinaison produit bien les fonctionnalités prévues.

— Test système : On s'assure que le système complet correspond bien à la spécification générale.

— Test de non-régression : C'est un cas particulier, où on vérifie que la correction des erreurs n'a pas affecté les parties déjà testées. (Cela consiste généralement en une réapplication systématique des tests déjà exécutés).

2. La paramètre de classification suivant est la méthode de test. On distingue deux grandes catégories :

— Le test structurel (ou white-box testing) : c'est une catégorie de test où le testeur a accès au code source et peut donc mettre en place des stratégies de test destinées à tester des parties spécifiques de ce code source (La suite de ce TD concernera les méthodes de test structurel)

— Le test fonctionnel (ou black-box testing) : c'est une catégorie de test où le testeur ne connait pas l'implémentation et ne peut donc proposer que des tests basés sur la spécification générale de l'objet à tester. Ils ont l'avantage de permettre de préparer les tests en amont de l'implémentation.

Il est possible d'effectuer ces tests de manière manuelle (déconseillé pour les gros projets), ou de manière automatisée ou de se placer dans des circonstances extrèmes (usage mémoire, performance,...)

2 Tests Structurels

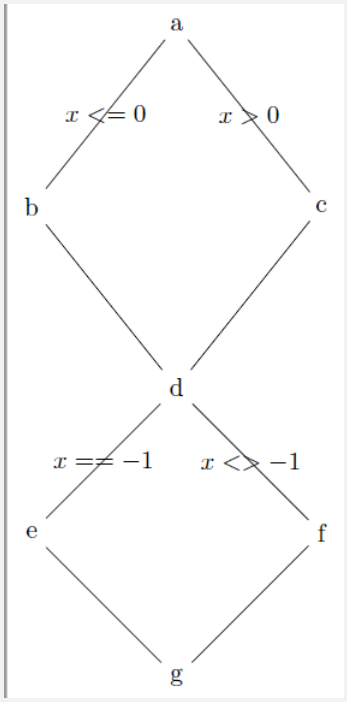
Graphes de Flot de Contrôle [Cours]

D'après ce que l'on vient de voir, les tests structurels sont ceux où le testeur a accès au code source de l'application qu'il doit tester. Nous allons nous servir de cette connaissance des sources pour créer des jeux de tests les plus pertinents possibles.
Pour cela, nous allons utiliser la notion de graphe de contrôle, qui va nous permettre de modéliser les différentes étapes du programme :

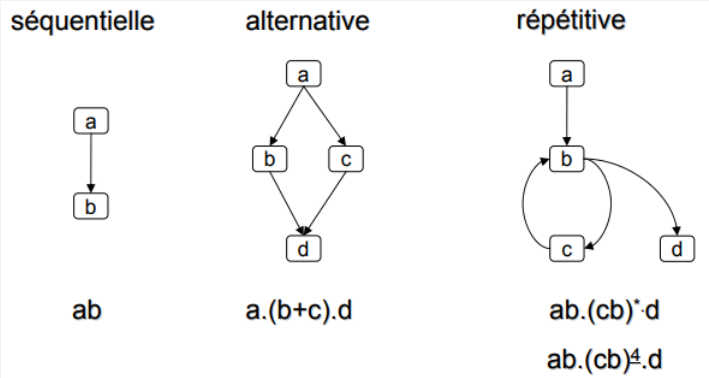
Un graphe de Flot de Contrôle est une représentation dous forme de graphe de tous les chemins qui peuvent être suivis dans le code source d'un programme lors de son exécution. Intuitivement, chaque sommet du graphe représente un bloc de code (une suite d'instructions simples, sans structure de contrôle), et chaque instruction de contrôle (if, boucle) crée une ou plusieurs flèche entre deux sommets. Par exemple, le programme suivant :

```
1  x=int(input())
2  if(x<=0):
3      x=-x
4  else:
5      x=1-x
6  print(x)
7  if(x==-1):
8      x=1
9  else:
10     x=x+1
11 print(x)
```

prend la forme du graphe :



On liste ensuite les chemins d'exécution possibles. Dans cet exemple, il s'agit de $abdeg, abdfg, acdeg, acdfg$. Notez qu'on peut condenser l'expression en utilisant des outils de factorisation : $a.(b+c).d.(e+f).g$. De manière plus générale, la construction des chemins d'exécution possibles peut se faire suivant les trois cas suivants :



L'objectif du test est alors de trouver des jeux de tests qui activent ces différents chemins, de manière à pouvoir s'assurer que toutes les parties du programme fonctionnent normalement.

Exercice 1 (, ⋆)
Proposez un jeu de test permettant d'activer tous les chemins du programme ci-dessus. □

Exercice 2 (, ⋆)
Soit le programme suivant :

```
def f(b,c,x):
    if b<c:
        d=2*b
         f=3*c
         if (x>=0):
              y=x
              e=c
              if(y==0):
```

```
                    a=f-e
                    if(d<a):
                          print(a)
                    else:
                          print(d)
```

1. *Donnez le graphe de flot de contrôle du programme.*
2. *Donnez trois chemins du graphe.*
3. *Donnez une expression générale pour tous les chemins du graphe.*
4. *soit le jeu de test $b = 1, c = 2, x = 2$. Quel chemin est activé par ce jeu ?*
5. *On s'intéresse aux deux prints. Donnez deux jeux de test qui vont couvrir ces instructions.*

□

Exercice 3 (, ⋆)

Soit le programme suivant :

```
def f(c):
    x=0
    if (c==1):
        x=x+1
    if (c==2):
        x=x-1
    return x
```

1. *Donnez le graphe de flot de contrôle du programme.*
2. *Donnez une expression générale pour tous les chemins du graphe. Combien y en a t'il ?*
3. *Certains chemins du graphe ne pourront jamais être exécutés. Donnez un exemple.*
4. *Les chemins non-exécutables indiquent en général un code peu fiable. Proposez une nouvelle version du code ci-dessous qui ne posera pas ces problèmes. Construisez son graphe de flot de contrôle et donnez une expression générale pour tous les chemins du graphe.*
5. *Proposez des jeux de test activant tous les chemins.*

□

Exercice 4 (, ⋆)

1. *Ecrivez un programme recherchant la position d'un élément donné dans une liste.*
2. *Donnez le graphe de flot de contrôle du programme.*
3. *Donnez une expression générale pour tous les chemins du graphe. Combien y en a t'il ?*
4. *En supposant des listes de taille 3, proposez des jeux de test activant tous les chemins.*

□

Les diverses stratégies de test structurel [Cours]

Trouver un ensemble de jeux de test couvrant tous les chemins du graphe, bien que souhaitable, est dans la pratique trop compliqué, voire impossible. On peut donc se rabattre sur des critères de test plus accessibles :

— On peut décicder de produire un ensemble de jeux de test couvrant tous les noeuds du graphe. Autrement dit, pour chaque sommet du graphe, il existe au moins un jeu de test l'atteignant.
Ce système a l"inconvénient de ne pas considérer comment sont atteints les sommets, et donc de potentiellement éviter des blocs de code.

— On peut décider de produire un ensemble de jeux de test couvrant tous les arcs du graphe. Autrement dit, pour chaque flèche du graphe, il existe au moins un jeu de test passant par cette flèche.
Ce système a l"inconvénient de ne pas considérer les interactions entre toutes les flèches, et donc entre tous les blocs de code.

Il en existe bien d'autres, plus ou moins efficaces pour capturer la complexité d'un programme.

On peut évaluer la qualité d'un jeu de test suivant ces deux critères en calculant le pourcentages de sommets (de flèches) atteint par le jeu. On appelle cette valeur le Test Effectiveness Ratio (TER).

Toutes les stratégies envisageables ont des défauts. Le choix d'une stratégie de test revient donc à peser le pour et le contre entre sécurité et faisabilité (technique ou budgétaire).

Exercice 5 (, $\star$)
Soit le programme suivant (où on suppose l composé de nombres strictement positifs) :

```
def f(l,inf,sup):
    res=0
    for i in range(inf,sup+1,1):
        res=res+l[i]
    return 1/res
```

1. *On choisit le jeu de test* $(l = [0, 1, 2], inf = 0, sup = 2)$. *que se passe-t-il ?*
2. *Calculez le TER de ce jeu de test pour les deux critères de couverture vus au-dessus. Qu'en pensez-vous ?*

□

Exercice 6 (, $\star\star$)
Trouver un exemple de programme pour lequel le critère de couverture des noeuds est insuffisant pour détecter une erreur. (Autrement dit, trouver un programme et un ensemble de jeux de test qui atteint bien tous les noeuds du graphe de ce programme, mais qui esquive de tester un aspect du programme). □