Projet : Exécution symbolique et tests structurels

Ayaz Badouraly & Venceslas Danguy des Déserts

Enseignante : Mme Pascale Le Gall

Table des matières

Chapitre 1

Architecture du projet

1.1 Pipe de traitement

Le code se divise en trois grandes parties : parsing du programme dans le langage WHILE, vérification de critères sur ce programme, génération des tests pour les critères.

1.1.1 Parsing

Le parsing consiste, à partir d'un programme écrit en langage WHILE, à obtenir un graphe de contrôle (CFG) représentant le dit programme. On passe pour cela par une étape intermédiaire : un arbre de syntaxe abstraite, ou AST. L'AST est une représentation du code du programme sous forme d'arbre, tandis que le CFG représente vraiment le flux du programme.

Génération de l'AST

Nous utilisons pour ce faire la librairie Python ply, qui fournit une implémentation en Python du duo lex/yacc. Lex tokenize le programme en entrée en fonction de la grammaire définie, tandis que yacc prend l'entrée tokenizée et fournit l'AST. Nous utilisons la librairie *anytree* pour manipuler les arbres.

Conversion de l'AST au CFG

Il faut ensuite traduire cet AST en CFG. On utilise pour cela un algorithme récursif qui va parcourir l'arbre et construire le CFG. Chaque exécution de la fonction récursive renvoie un morceau de graphe ainsi que des arêtes sortantes à relier.

Ainsi, la conversion d'un arbre ne contenant en tout et pour tout qu'une assignation va renvoyer un noeud labellisé et une arête sortante de ce noeud portant une condition *True* et une commande *Assign*.

La conversion d'un arbre contenant un noeud While va relier les arêtes sortantes du graphe du sous-arbre au noeud parent de façon à former une boucle, etc.

On ajoute enfin un tag sur les noeuds correspondant à des instructions *If* ou *While* pour faciliter la vérification et la génération des tests.

Les graphes sont gérés par la librairie python *networkx*.

1.1.2 Vérification de critères

Nous sommes donc désormais capables d'obtenir le CFG associé à un programme écrit en langage WHILE. Nous voulons maintenant *vérifier* des critères de tests pour ce programme, c'est à dire, étant donné un jeu de tests (données de départs), dire si les tests couvrent un ensemble de cas (chemins d'exécution) précis.

Il faut donc commencer par exécuter le programme avec le jeu de test. Ceci se fait assez facilement, il suffit de parcourir le graphe de contrôle en maintenant un état (ensemble des variables et de leurs valeurs) et en choisissant, à chaque branchement, l'arête dont la condition est vérifiée par l'état.

On obtient ainsi un ensemble de chemins d'exécution (un par valuation de départ).

Parallèlement, on a un critère de test, c'est à dire un ensemble de chemins par lesquels nous voulons passer. Ainsi le critère *Toutes les affectations* demande-t-il qu'à la fin de l'exécution de tous les tests, on soit passé au moins une fois par chaque arête portant une commande *Assign*.

Chaque critère possède son propre algorithme de génération d'éléments à vérifier en fonction du programme (bien que l'on puisse toujours se ramener à un ensemble de chemins, il était parfois plus simple de considérer un ensemble de noeuds ou d'arêtes par lesquels il faut passer).

Il suffit ensuite de faire l'intersection des deux ensembles *{Chemins exécutés lors de la phase du test}* et *{Chemins à vérifier imposés par le critère}*.

1.2 Choix de conception

1.3 Limitations adoptées

1.4 Organisation du code

Le code est organisé en modules.

- **astree :** Contient notre structure d'AST avec les commandes et opérations définies par la sémantique opérationnelle de WHILE.
- **cfgraph :** Code utilisé pour exécuter un test sur le programme. Le module utils, en particulier, contient toutes les fonctions du graphe nécessaires pour générer les éléments à vérifier pour chaque critère.
- **syntax :** Contient le code du lexeur et du parseur ainsi que leur configuration
- **tests :** Ce dossier contient le coeur du projet. Le module *tester* contient la logique de vérification des critères : récupération des éléments à vérifier, analyse des chemins d'exécutions des tests pour y trouver les éléments. Le module *generator* gère la génération des jeux de tests en faisant appel au solveur *Z3*.
- **utils :** Contient notamment la conversion de l'AST vers le CFG.
- **input :** Contient des exemples de programmes sur lesquels on peut vérifier les critères.
- **output :** Exemples de graphes (images).

Chapitre 2

Vérification des critères et génération des tests

2.1 Toutes les affectations

On va travailler sur le programme suivant :

```
1:  if (X <= 0) {
2:      X := -X;
    } else {
3:      X := 1 - X;
    }

4:  if (X == 1) {
5:      X := 1;
    } else {
6:      X := X + 1;
    }
```

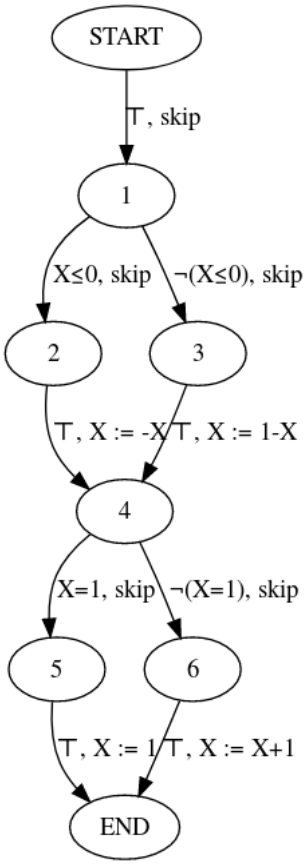

FIGURE 2.1 – CFG associé

Le critère *Toutes les affectations* demande à ce que tous les labels d'affectations (ici : $2, 3, 5, 6$) apparaissent au moins une fois dans l'un des chemins d'exécution correspondant aux données de tests.

Si on prend pour jeu de test :

$$\{\{X : -1\}, \{X : 1\}\}$$

On obtient les chemins d'exécutions suivants :

$\{X : -1\}$: $1, 2, 4, 5$

$\{X : 1\}$: $1, 3, 4, 6$

On est donc passé au moins une fois par chaque label associé à une affectation : le critère est vérifié sur ce programme et ce jeu de test.

Si, en revanche, on s'était contenté d'un unique test $\{X : -1\}$, on ne serait pas passé par les noeuds 3 et 6, et le critère n'aurait pas été vérifié.

Dans la pratique, étant donné un CFG, on récupère l'ensemble des noeuds (labels) tels qu'une arête sortante de ce noeud porte une affectation. Ceci se fait

très facilement en parcourant l'ensemble des arêtes (sans notion de chemin ici). Il suffit ensuite de regarder lesquels de ces noeuds sont (ou non) dans les chemins des tests.

2.2 Toutes les décisions

On travaille toujours sur le programme défini en 2.1.

Le critère *Toutes les décisions* demande à ce que l'on passe au moins une fois par toutes les arêtes associées à une expression booléenne. Il s'agit, en bref, de toutes les arêtes sortantes des noeuds *if* et *while*, ce qui est équivalent à dire qu'il faut passer par les noeuds de destination de ces arêtes.

Ici, il faut donc passer par les noeuds $2, 3, 5, 6$. On retombe dans ce cas très simple sur les même éléments que le critère *Toutes les affectations*, et le même jeu de tests vérifie donc les deux critères.

2.3 Tous les k-chemins

2.4 Toutes les i-boucles

2.5 Toutes les définitions

Le critère *Toutes les définitions* demande que, pour chaque variable X, pour chaque noeud u affectant X, on ait un chemin $..u...v..$ tel que v utilise X, et qu'il n'y ait pas de réaffectation de X entre u et v.

Pour vérifier ce critère, nous commençons par implémenter les fonctions *def* et *ref* telles que définies dans l'énoncé.

On construit ensuite un dictionnaire *definitions* qui associe à chaque variable X l'ensemble des noeuds u tels que $X \in def(u)$.

Pour chaque variable X, on regarde ensuite chaque chemin. Si on trouve un noeud u intéressant, c'est à dire $X \in def(u)$, on suit le chemin jusqu'à trouver v tel que $X \in ref(v)$. On s'assure bien sûr qu'entre u et v, il n'y a pas de noeud u' tel que $X \in ref(u')$. Si on trouve v, on supprime u de *definitions*$[X]$.

Si à la fin, *definitons* est vide, alors le critère est vérifié.

Le programme de la partie 2.1 ne pourra jamais vérifier ce critère, quel que soit le jeu de test, car les affectations 5 et 6 ne sont jamais utilisées. Si on rajoute une commande print qui affiche une variable, ainsi qu'une nouvelle variable Y, on peut le réécrire :

```
1:  if (X <= 0) {
```

```
2:      X := -X;
3: Y := 5;
    } else {
4:      X := 1 - X;
    }

5:  if (X == 1) {
6:      X := 1;
    } else {
7:      X := X + 1;
8: print(Y);
    }
9: print(X);
```

Le dictionnaire *definitions* s'écrit alors :

$$\{X : [2, 4, 6, 7], Y : [3]\}$$

Le jeu de tests suivant ne vérifie pas le critère :

$$\{\{X = 1, Y = 1\}, \{X = -1, Y = 1\}\}$$

(Pas de chemin entre 3 et 8)
Contrairement à cet autre jeu de tests :

$$\{\{X = 1, Y = 1\}, \{X = -2, Y = 1\}\}$$

2.6 Toutes les utilisations

2.7 Tous les DU-chemins

2.8 Toutes les conditions