TP Test Logiciel

L'objectif de ce TP est d'utiliser les jeux de tests générés par des outils automatiques de couverture de code pour trouver et expliquer des erreurs dans différents programmes à tester. Un premier aspect du travail sera de définir les contours des domaines de test à explorer, pour faciliter la génération de tests faisant apparaître les éventuelles erreurs. Un deuxième aspect sera l'écriture d'oracles, c'est-à-dire de petits programmes annexes qui traduisent les post-conditions des programmes à tester et qui seront appelés pour vérifier le résultat des programmes à tester. Les fichiers des programmes à tester et les squelettes des oracles à compléter sont disponibles à l'adresse : `/home/MEEA/TP_test`

Pour cette séance, nous utiliserons la version en ligne de l'outil PathCrawler, à l'adresse suivante : `http://www.pathcrawler-online.com`

Important Pour ménager le serveur web distant, il vous est demandé d'avoir à chaque instant un unique onglet ouvert à cette adresse.

Travail Vous devrez rendre un compte rendu avec les fichiers des oracles complétés. Le rapport comprendra les réponses aux questions du TP (qui tiennent en général sur une ligne), et le détail sur certains tests générés par PathCrawler. La description d'un test indiquera le fichier testé, le domaine des paramètres du test, le verdict, et zéro, un ou deux cas jugés intéressants parmi les tests générés (pour ceux-ci, donner les entrées et sorties, le chemin, et la condition de chemin, et indiquez en quoi ils sont intéressants). Quand vous diagnostiquez une erreur, vous pouvez la désigner par sa ligne et sa nature. Que ce soit par un tableau ou un texte brut, présenter ces données sous une forme lisible fait partie de l'exercice.

Connectez-vous sur la machine avec votre identifiant (m2istr_xx) et mot de passe (jsupe2008). Un éditeur de texte (Gedit, Emacs, etc.) sera utilisé pour éditer des fichiers.

Pour prendre contact avec PathCrawler, nous utiliserons les exemples de la racine carrée entière et de la recherche dans un tableau. Il y a l'exemple du tri de tableau si vous avez le temps !

Pour générer les données de test, vous devrez vous aider des graphes de flow de contrôle (CFG) des implémentations. Bien sûr, il vous revient de construire ces CFG !

1 Racine carrée entière

Voici la spécification utilsée pour les tests :

> La fonction de racine carrée entière *sqrt*() prend en argument un entier a et renvoie l'unique entier i tel que $i^2 \leq a < (i+1)^2$ si a est positif, et 0 sinon.

Génération de tests Dans le dossier `sqrt`, les fichiers `sqrt*.c` contiennent chacun une implémentation de la fonction `sqrt` (dont certaines contiennent des erreurs), et le fichier `oracle_sqrt.c` contient le squelette d'un oracle.

Pour les envoyer sur le serveur de PathCrawler, il faut les placer dans une archive zip, en invoquant la commande suivante (en étant placé dans le dossier `sqrt` toujours) : `zip sqrt.zip *.c`

Cette manipulation sera à refaire après toute modification de l'un des fichiers. Commencez par lancer la génération de tests sur la fonction du fichier `sqrt1.c`. Pour ceci, allez dans l'onglet "Test your code", puis :

1. Téléchargez l'archive zip dans la case "Upload archive".
2. Indiquez dans la case "Test function" le nom de la fonction à tester (ici `sqrt`).
3. Indiquez dans la case "File under test" le fichier dans lequel se trouve la fonction à tester (ici `sqrt1.c`).
4. Cliquez sur le bouton "Customize test parameters" pour définir le domaine du test. Vous pourrez notamment fixer les limites dans lesquelles l'argument a sera choisi.
5. Quand tout est renseigné, cliquez sur un bouton "Run test".

Après un petit temps d'attente, vous arrivez à un écran donnant un bilan des tests qui ont été effectués.

— L'onglet "Session" affiche un résumé général.
— L'onglet "Test cases" vous permet de voir le détail de chaque test généré, avec en particulier les valeurs d'entrée et de sortie, le chemin emprunté dans le programme, et la condition de chemin correspondante. Pour l'instant, tous les cas de test devraient être dans une case orange ("unknown"), indiquant que l'oracle n'a pas dit si le résultat était ou non correct.
— L'onglet "Path" donne tous les chemins qui ont été explorés dans le programme, y compris ceux qui n'ont pas généré de test, par exemple en raison d'une condition de chemin incohérente (ou "infaisable", pastille bleue).
— L'onglet "Context" regroupe les informations de votre session (programme testé, domaine du test, oracle).
— L'onglet "Restart" vous permet de relancer la génération de tests pour le même programme à tester, éventuellement en modifiant le domaine des tests.

Question 1 Allez consulter les tests qui ont été générés. Observez les valeurs d'entrée et les conditions de chemins qui ont été calculées. Allez également consulter l'onglet "Path", qui montre qu'un des chemins explorés était infaisable. À quoi correspondrait ce chemin ?

Question 2 Répétez l'opération avec la fonction `sqrt` du fichier `sqrt2.c`. Comparez les chemins explorés avec ceux de `sqrt1.c` et expliquez la différence en allant voir le code pour.

Question 3 Testez les fichiers `sqrt1.c` et `sqrt3.c`, avec un domaine pour a allant de 0 à 25. Que remarquez-vous ? Quelle erreur est à l'origine de ce phénomène ?

Ecriture d'oracle L'oracle est un petit programme C qui peut consulter les entrées et sorties du programme à tester. Il est conçu uniquement en fonction des spécifications et de la signature de la fonction à tester.

Ainsi, nous écrirons un seul oracle `oracle_sqrt.c`, qui sera utilisé pour chacune des implémentations de la fonction `sqrt`.

Dans l'oracle, la variable `a` donne accès à l'entrée `a` du programme à tester, et la variable `pathcrawler_retres_sqrt` donne accès au résultat du programme à tester.

Avant le `return` final, pour indiquer que le résultat du programme à tester est correct ou incorrect, il est possible d'invoquer respectivement l'une des fonctions :

`pathcrawler_verdict_success()` ou `pathcrawler_verdict_failure()`

Question 4 Rappelez la propriété que doit vérifier le résultat de la fonction `sqrt` et complétez l'oracle dans le fichier `oracle_sqrt.c`.

Question 5 Testez toutes les implémentations fournies pour la fonction `sqrt`.

La génération de tests comportant une part d'aléatoire, il peut être intéressant de lancer plusieurs fois les tests, et de varier le domaine des tests pour observer le plus de choses possibles.

Pour chaque implémentation, donnez dans votre rapport un exemple d'un jeu de tests générés par PathCrawler, indiquez les cas qui ont révélé une erreur, et allez consulter le code source pour expliquer ces erreurs.

2 Recherche dans un tableau trié

Nous nous plaçons maintenant dans le dossier `search`, qui contient également plusieurs fichiers `search*.c` proposant chacun une implémentation d'un programme `search` de recherche d'un élément dans un tableau trié, et un fichier `oracle_search.c` dans lequel vous devrez écrire un oracle.

Notre fonction `search` prend en argument un tableau `A` et un élément `elem`, et renvoie l'index auquel `elem` est présent dans le tableau `A` s'il y est présent, et -1 sinon. Pour les tests, nous avons fixé la taille des tableaux à 8.

Question 1 L'implémentation `search1.c` utilise une recherche linéaire, et `search2.c` une recherche dichotomique. Comparez le nombre et la taille des chemins explorés par ces deux implémentations.

Question 2 Ajoutez une précondition pour indiquer que le tableau d'origine doit être trié. Cela se fait dans la définition du domaine des tests, à la ligne "Quantified preconditions" (pour quantifier sur les indices du tableau, il faudra nommer la variable quantifiée `INDEX`). En testant à nouveau `search1.c` ou `search2.c`, avec cette restriction du domaine, vous pourrez observer que certains cas de test ont disparu.

Lequel de ces deux fichiers est concerné ?

Tâchez de trouver un cas de test qui a été éliminé par la précondition, et d'expliquer comment.

Question 3 Donnez la spécification de la fonction de recherche.

Complétez l'oracle du fichier `oracle_search.c` pour traduire cette spécification.

Question 4 Testez toutes les implémentations comme à la partie précédente, en variant le domaine de test choisi. Pour chaque fichier dans lequel une erreur est révélée, donnez un exemple de test ayant montré cette erreur et un exemple de test qui reste correct. Consultez le code source pour expliquer ces erreurs.

Quand cela est pertinent, vous pouvez aussi proposer une modification des préconditions du programme qui élimine les cas d'erreurs. N'oubliez pas de toujours renseigner la précondition quantifiée imposant au tableau d'être trié.

Question 5 Lancez un test sur `search1.c` sans remplir la précondition qui impose au tableau d'être trié. Probablement, tous les voyants seront au vert, ce qui est anormal.

Donnez un cas concret d'entrée où le programme ne trouve pas un élément qui est pourtant présent dans un tableau non trié. Jouez sur le domaine du test et/ou sur les préconditions pour provoquer l'apparition d'un tel contre-exemple (n'oubliez pas que cela peut ne pas marcher dès le premier essai en raison du caractère aléatoire des tests choisis).

Précisez dans votre rapport les conditions qui auront permis à l'erreur d'apparaître, ainsi que le test généré correspondant.