Génie Logiciel Avancé

Année 2021-2022

Prof. Burkhart Wolff
wolff@lri.fr

Hugo Mlodecki, Carmelo Vaccaro
hugo.mlodecki@universite-paris-saclay.fr
carmelo.vaccaro@universite-paris-saclay.fr

TP - Test en boîte blanche avec PathCrawler

Semaine du 8 novembre 2021

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 :

`http://www.lri.fr/~blsk/PathCrawler.zip`

Pour cette séance, nous utiliserons la version en ligne de l'outil PathCrawler :

`http://pathcrawler-online.com:8080`

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

Exercice 1 (Racine carrée entière)

Après avoir décompressé l'archive des fichiers de ce TP, placez-vous dans le dossier `sqrt`.

La fonction de racine carrée entière `sqrt` prend en argument un entier `a` et renvoie l'unique entier `i` tel que $\texttt{i}^2 \leq \texttt{a} < (\texttt{i}+1)^2$ 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 : limitez le test avec les conditions $0 \leq \texttt{a} \leq 100$.
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 générés puis exécutés.

— 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 (*input values* et *output values*), le chemin emprunté dans le programme, et la condition de chemin correspondante (*path predicate*). 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 A. 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 ?

Au lieu de contraindre les valeurs possibles de `a`, changez la stratégie de génération (par défaut *all-path*, tous les chemins) par *5-path* (tous les chemins passant au plus k fois dans les boucles, avec $k = 5$). Y a-t-il encore des chemins infaisables ? Pourquoi ?

Question B. Générez des tests pour la fonction `sqrt` du fichier `sqrt2.c`, avec les mêmes valeurs possibles pour `a`. Comparez les chemins explorés avec ceux de `sqrt1.c` et expliquez la différence (vous avez le droit d'aller voir le code pour ceci !).

Question C. 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 ?

Écriture d'oracles. 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, il est possible d'invoquer l'une des fonctions `pathcrawler_verdict_success()` ou `pathcrawler_verdict_failure()`, respectivement pour indiquer que le résultat du programme à tester est correct ou incorrect.

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

Question E. 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, trouvez les erreurs le cas échéant et allez consulter le code source pour expliquer ces erreurs.

Exercice 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`, un entier `n` (la taille du tableau) et un élément `elem`, et renvoie l'indice auquel `elem` est présent dans le tableau `A` s'il y est présent, et -1 sinon.

Question A. Formalisez les pré-conditions de cette fonction. Utilisez la personnalisation des paramètres de test pour que ces pré-conditions soient prises en compte pour la génération. On utilisera les lignes Unquantified preconditions et Quantified preconditions. Pour quantifier sur les indices du tableau, il faudra nommer la variable quantifiée `INDEX` (ou `INDEX_n` pour `n` valant `0`, `1`, `2`...). Il faudra également restreindre la taille possible de `A` pour générer un nombre raisonnable de tests (par exemple en la fixant à 10). Il pourra être nécessaire aussi de restreindre les valeurs possibles pour les éléments de `A` et pour `elem` pour faire apparaître certaines fautes.

Question B. 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 C. Exprimez en langage logique la post-condition de la fonction de recherche. Complétez l'oracle du fichier `oracle_search.c` pour traduire cette spécification. Attention, un appel à une fonction de verdict doit nécessairement être suivi d'un `return`.

Question D. Testez toutes les implémentations comme à la partie précédente, en variant le domaine de test choisi. Consultez le code source pour expliquer les erreurs trouvées. N'oubliez pas de toujours renseigner les pré-conditions dans les paramètres de test.

Question E. 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).

Exercice 3 (Tri d'un tableau)
Nous nous plaçons maintenant dans le dossier `sort`, qui est structuré comme les précédents. La fonction `sort` prend en argument un tableau, dont elle doit trier les éléments en place.

Question A. Exprimez en langage logique la spécification de la fonction de tri, puis complétez l'oracle du fichier `oracle_sort.c` en conséquence. Indication : dans l'oracle, le tableau `Pre_table` correspond à l'état du tableau avant le tri, et le tableau `table` correspond à son état après le tri.

Question B. Testez toutes les implémentations fournies, avec une taille inférieure ou égale à 5 pour le tableau `table`. Comme dans l'exercice précédent, vous étudierez les différentes erreurs révélées et chercherez à expliquer ces erreurs.