Saphir Gobbi, Aude Mazet

APNEE2 ALGO

Introduction : (les 500 programmes à tester sont dans le dossier all_programs, il est à mettre dans le répertoire courant d'où on va lancer le programme check.sh et test1.c, ainsi que tous les autres fichiers d'en-tête. Un Makefile est fournie).

Dans ce travail, nous avons été amenés à concevoir et implémenter des algorithmes de génération automatique de chaînes de caractères aléatoires en nous concentrant sur leur parenthésage. Plus précisément, nous devions d'abord créer deux fonctions capables de générer des chaînes bien parenthésées, avant de les adapter pour générer des chaînes mal parenthésées. Pour ce faire, nous avons choisi de nous concentrer sur tous types de parenthèses s, c'est-à-dire «(» et «)», «{» et «}», «[» et «]», selon notre interprétation de la consigne.

Le but était de tester ces chaînes sur des programmes de vérification de parenthésage, afin de détecter d'éventuels bugs.

Pour avoir les informations concernant l'utilisation du programme test1.c, il suffit de lancer le programme avec la commande : ./test1 afin d'obtenir les informations nécessaires concernant son utilisation.

La chaine générée par test1 sera stockée dans output.txt.

Tous les résultats des tests seront stockés dans result.txt par programme testé.

Exercice 1 :

Dans la première fonction generer_parenthesage_aleatoire_lg, nous avons décidé d'ajouter un caractère tant que la longueur minimum de la chaine passée en paramètre n'était pas atteinte et tant que la pile qui nous sert à garder le compte des parenthèses n'était pas vide. Avec cette condition de boucle, on s'assure que la longueur minimale est respectée et on s'assure que la chaine est correctement parenthésée. (En effet, quand la pile est vide, cela signifie qui y a autant de parenthèses fermantes que d'ouvrantes.) Pour les caractères à imprimer nous avons utilisé « rand() % 3 » qui nous permet un tirage de 0 à 2. Ainsi chaque numéro correspond à un caractère à imprimer à condition que les prérequis soient respectés (par exemple, on n'imprime une parenthèse fermante que si la pile n'est pas vide pour ne pas créer d'erreur), ce qui rend la chaine parfaitement aléatoire. Nous avons suivi ce procédé également pour la deuxième fonction demandée, ainsi generer_parenthesage_aleatoire_imb imprime un caractère aléatoire grâce à rand() tant que le niveau d'imbrication n'est pas atteint.

Nous avons testé nos programmes avec les valeurs 20, 50, 100, 300, 500, 1000, 2000 afin d'avoir un bon aperçu des résultats obtenus sur une grande échelle.

Ainsi pour identifier les bugs nous avons généré plusieurs chaines aléatoires bien parenthésées avec différents arguments, nous nous attendions donc à avoir nos programmes testés qui renvoient « Bon parenthesage », ainsi ceux qui renvoyaient autre chose présente un bug évident.

Le nombre de programmes incorrects par chaines générées est stockée dans le fichier false_result.txt.

Exercice 2 :

Pour générer une chaine mal parenthésée nous avons pris en compte plusieurs cas, nos chaines aléatoires doivent présenter tous types d'erreurs, comme avoir une ou plusieurs parenthèses (ou ses variantes) ouvrantes en trop ou avoir une ou plusieurs parenthèses (ou ses variantes) fermantes en trop, de plus elles doivent également présenter des cas de parenthèses (ou ses variantes) mal imbriquées, comme «)(« ou une chaine commençant par «) » ou finissant par « (».

Notre premier algorithme génère une chaîne de caractères avec un parenthésage incorrect d'une longueur minimale spécifiée. Il commence par déterminer aléatoirement combien de parenthèses incorrectes seront ajoutées avant et après une portion correctement parenthésée. Ensuite, une première partie de texte est générée avec des parenthèses mal placées, suivie d'un appel à une fonction qui génère un bloc de texte correctement parenthésé. Enfin, une seconde partie avec des parenthèses incorrectes est ajoutée. Le but est de créer un texte globalement mal parenthésé tout en respectant la longueur minimale.

Nous avons suivi le même raisonnement pour créer le deuxième algorithme qui suit donc ce même schéma.

De plus, pour générer des erreurs de parenthésages spécifiques nous avons créé des fonctions complémentaires. Par exemple, les fonctions generer_mauv_no_inc et generer_mauv_no_dec simulent respectivement des erreurs où les parenthèses ouvrantes ne sont pas comptées et où les parenthèses fermantes ne sont pas prises en compte, tout en respectant la longueur minimum. De même, la fonction generer_mauv_no_type génère des chaînes qui ne font pas la distinction entre les différents types de parenthèses, ce qui peut tromper un programme qui devrait normalement les traiter différemment. Enfin, la fonction generer_mauv_no_succeder crée des chaînes où les parenthèses ouvrantes sont comptées jusqu'à la première parenthèse fermante rencontrée, après quoi les ouvrantes ne sont plus prises en compte. Ces fonctions ont été conçues pour tester la robustesse des programmes de vérification de parenthésage de façon plus précise.

De façon opposé à l'exercice 1, tous les programmes renvoyant autre chose que « Mauvais parenthesage » présentent un bug.

Pour cet exercice nous avons gardé les valeurs des paramètres de l'exercice 1, nous avons donc testé nos programmes avec les valeurs 20, 50, 100, 300, 500, 1000, 2000.

Nous avons généré 64 chaines aléatoires en comptant la chaine vide.

Explication de check.sh :

On fait des tests, qui seront effectués sur les 500 programmes, pour chaque chaine générée. Puis on les met dans des fichiers temporaires. A la fin des tests, on archive les résultats par programmes, dans un fichier « result.txt », et on archive également dans le fichier « false_result.txt » le nombre de programmes qui ont eu une mauvaise réponse par chaine.

Ensuite, on archive toutes les chaines générées qui ont été testé par les 500 programmes, et on les transferts dans un dossier du répertoire courant "all_chaines/" (Si il n'existe pas déjà il sera créé).

Et enfin, on supprime tous les fichiers temporaires.

Exercice 3 :

Nous avons ainsi implémenté la fonction de l'oracle qui nous a permis de vérifier quels programmes renvoyaient un résultat diffèrent de celui attendu.

Conclusion :

Ce projet a permis de développer des algorithmes efficaces pour générer des chaînes correctement et mal parenthésées, avec des techniques de programmation aléatoire basées sur l'algorithme de vérification de parenthésage. Les fonctions spécifiques créées pour introduire des erreurs nous ont été utile pour tester la robustesse des programmes fournis. Grâce à une série de test, nous avons identifié plusieurs bugs dans les programmes. Ces résultats montrent que la génération automatisée de tests est une méthode efficace pour améliorer la qualité et la fiabilité des logiciels.