Utilisation des Structures Combinatoires pour le Test Statistique

Sandrine-Dominique GOURAUD

Le test de logiciel

Du cahier des charges à l'intégration du logiciel, le processus de création des logiciels est long et complexe. Pour assurer que le produit est conforme aux objectifs attendus, il faut valider le résultat du développement. En effet, dans le cas des logiciels critiques, comme ceux qui dirigent les centrales nucléaires ou encore les systèmes embarqués des avions, trains et autres transports, les conséquences d'une erreur peuvent être dramatiques car ce sont des catastrophes humaines, environnementales ou économiques qui sont en jeu. Les contraintes de fiabilité de ces logiciels sont très importantes, c'est pourquoi des chercheurs, comme des industriels, travaillent au développement de méthodes efficaces et surtout d'outils pour automatiser les activités de vérification et de validation des logiciels, et notamment leur test.

Le test de logiciels n'est pas un procédé facile à mettre en place. Les méthodes de test progressent rapidement, mais en parallèle, on en demande de plus en plus aux logiciels et leur complexité augmente. En mai 2002, une étude[1] menée pour le NIST sur l'impact de l'insuffisance d'infrastructure de test dans les développements de logiciels a montré que le coût de cette insuffisance est d'environ 60 milliards de dollars pour l'économie américaine. Il faut donc développer de nouvelles méthodes de tests automatisables.

Les différents types de test Dans le processus de développement du logiciel, il existe différents types de test dont: les tests unitaires qui permettent de tester les composants un par un, les tests d'intégration qui permettent de tester un ensemble de composants, les tests "système" qui permettent de tester le système sur son futur site d'exploitation dans des conditions opérationnelles et au-delà[2] (surcharge, défaillance matérielle, etc.), et les tests de régression qui permettent de vérifier que la modification d'un ou de plusieurs composants n'entraîne pas d'incohérence entre la nouvelle version et la précédente. Dans ce mémoire, nous nous sommes intéressés aux tests unitaires.

La méthode la plus connue pour tester un logiciel consiste à sélectionner des entrées caractéristiques pour ce logiciel et à vérifier que le résultat et/ou le comportement obtenu

1. Étude menée auprès de plus de 200 acteurs (banques et industriels) pendant 18 mois.
2. Dans ce cas, le terme de test de robustesse est généralement employé.

est conforme à ce qui était attendu. Il existe de nombreuses méthodes pour sélectionner des ensembles de test. Classiquement, on répartit ces méthodes en trois familles: les méthodes *fonctionnelles* qui sélectionnent l'ensemble des entrées à partir de la spécification du programme à tester, les méthodes *structurelles* qui sélectionnent cet ensemble à partir de son code (instructions, enchaînements, chemins d'exécution, etc.) et les méthodes *statistiques* qui sélectionnent cet ensemble à partir d'une distribution que doivent satisfaire les données en entrée.

La qualité d'un jeu de test peut s'évaluer à sa capacité à couvrir un critère de couverture. Parmi les critères de couverture les plus connus, on trouve "tous les chemins", "tous les sommets (resp. arcs)" du graphe représentant le code d'un programme ou sa spécification, etc. Dans le cas d'une méthode de test statistique, cette qualité est liée à la probabilité minimale d'être atteint qu'a un élément du critère.

Une autre technique pour évaluer un jeu de test consiste à mesurer sa capacité à détecter des mutants (clones du programme de référence dans lesquels on a injecté une erreur). Pour un jeu de tests donné, le taux de mutants détectés est appelé score de mutation. Plus ce score est proche de 1 plus le jeu de test est pertinent. On valide alors à la fois le jeu de test et la méthode qui l'a produit.

Le test statistique ou aléatoire Le principe du test statistique est de tirer des données dans le domaine d'entrée selon une certaine distribution. Dans sa version classique, la distribution est uniforme: on ne tient compte ni de la spécification ni du code du programme testé. La distribution des données d'entrée peut également être estimée en fonction de l'exploitation future du système à tester. On parle alors de test aléatoire opérationnel.

Le test aléatoire uniforme permet d'assurer une bonne couverture des programmes dont les chemins d'exécution[3] ont des domaines d'entrée de tailles (et donc de probabilités) comparables, mais dans le cas contraire, il doit être complété notamment par des tests aux limites. En effet, il existe des cas particuliers (comme les cas d'exception) qui correspondent à des sous-domaines d'entrée de probabilité trop faible.

L'avantage des méthodes de test statistique est qu'à partir du moment où cette méthode est automatisée, elle permet de faire du test intensif. Lorsque que la distribution est uniforme et le domaine d'entrée est composé d'intervalles numériques, il est facile d'engendrer des jeux de test.

Contrairement aux approches classiques qui se basent sur une distribution sur les entrées, nous nous sommes intéressés aux distributions sur les chemins d'exécution ou des traces[4] du système à tester. Notre approche est donc mixte. Elle combine le test statistique avec le test structurel si l'on s'intéresse aux chemins d'exécution, ou le test fonctionnel si l'on s'intéresse aux traces.

3. Un chemin d'exécution d'un programme est la suite d'instructions exécutée pour les valeurs d'entrée considérées.

4. Une trace est une séquence d'entrées/sorties du système. Cela correspond à un chemin de l'automate à entrées et sorties qui modélise une spécification du système à tester.

Test de logiciel et structures combinatoires

L'étude et la simulation des processus stochastiques a largement profité des progrès dans le domaine de la génération aléatoire de structures combinatoires. Une structure combinatoire est soit un objet atomique, soit un objet obtenu en combinant à l'aide d'opérateurs des objets atomiques.

Les travaux de Wilf et Nijenhuis ont amené à l'élaboration d'algorithmes efficaces pour la génération aléatoire et uniforme d'une variété de structures combinatoires. En 1994, Flajolet, Zimmermann et Van Cutsem ont généralisé et systématisé cette approche en s'appuyant sur une décomposition récursive non ambiguë des structures combinatoires à engendrer. Leurs travaux constituent la base d'un outil puissant de génération uniforme d'objets complexes comme des graphes, des arbres, des mots, des chemins...

Dans cette thèse, nous explorons l'usage de tels concepts et outils pour le test de logiciel. En effet, nous proposons une méthode pour le test statistique basée sur le tirage de chemins dans un graphe en utilisant la génération aléatoire de structures combinatoires.

Principe général Notre approche consiste à utiliser les outils de génération de structures combinatoires à partir d'un graphe G représentant les comportements du système à tester. Cette description peut être le graphe de contrôle ou de flots de données du programme, une représentation graphique du système comme un réseau de Pétri, un Statechart... ou encore un système de transitions, un automate, une machine d'états, une structure de Kripke, etc. Toutes ces descriptions peuvent être décrites sous forme d'une structure combinatoire étiquetée.

Soient D une description d'un système à tester, C un critère de couverture, $E_C(D)$ un ensemble d'éléments de D satisfaisant C et N un nombre de tests à générer. La première étape de notre approche consiste à tirer un ensemble de N chemins telle que la probabilité minimale qu'a un élément de $E_C(D)$ d'être atteint soit maximale. Si $E_C(D)$ dénote directement un ensemble fini de chemins de G, comme dans le cas du critère "tous les chemins de longueur inférieure ou égale à n", la qualité de test est optimisée si les chemins de $E_C(D)$ sont tirés uniformément.

Par contre, si $E_C(D)$ n'est pas décrit comme un ensemble de chemins mais un ensemble d'éléments constitutifs de G (des sommets, des arcs, des circuits, etc.) nous devons alors maximiser la probabilité minimale d'atteindre un élément lors du tirage des chemins, ce que ne garantit pas la génération uniforme de chemins sur laquelle nous nous appuyons.

Le tirage des chemins devient plus compliqué et nous proposons de décomposer ce tirage en deux étapes:

1. choisir aléatoirement N éléments e_1, ..., e_N de $E_C(D)$, selon une distribution adéquate des probabilités;

2. pour chaque élément e_i tiré, tirer uniformément un chemin de G parmi tous ceux passant par e_i.

Dans cette thèse, nous avons proposé trois distributions différentes sur les éléments: une distribution uniforme, une distribution basée sur la notion de "dominance" dans un graphe et une distribution déterminée par la résolution d'un système d'équations qui permet de maximiser la probabilité minimale d'atteindre un élément. La distribution uniforme n'étant pas satisfaisante en pratique, seules les deux dernières ont été utilisées pour les expériences.

Une fois l'ensemble des N chemins obtenu, l'étape suivante consiste à déduire les valeurs d'entrées qui permettront de parcourir exactement les chemins sélectionnés lorsque cela est possible (certains chemins peuvent ne correspondent à aucune exécution). On construit pour chaque chemin le prédicat sur les valeurs d'entrée qui provoquent l'exécution de ce chemin. Puis pour déterminer des valeurs d'entrée qui permettront d'exécuter ces chemins, on résoud les prédicats. Malheureusement, dans le cas général, la résolution de prédicat est un problème indécidable. On s'appuie alors sur des techniques de résolution de contraintes pour essayer de le faire.

Une fois la résolution de tous ces prédicats effectuée, une analyse des résultats permet de trier ces prédicats en trois catégories:

1. ceux qui ont une solution: c'est le cas le plus simple. La solution est la donnée de test qui permet d'exécuter le chemin associé.
2. ceux qui n'ont pas de solution: le chemin associé est prouvé infaisable
3. ceux dont on ne sait pas s'ils ont ou non une solution

Pour les deux derniers cas, on effectue autant de nouveaux tirages qu'il y avait de prédicats sans solution.

Prototype et expériences réalisées Notre approche a été appliquée au test statistique structurel: la description utilisée est le graphe de contrôle et les critères de couverture étudiés sont "tous les chemins", "tous les enchaînements" et "toutes les instructions". Un prototype a été développé afin d'étudier expérimentalement l'efficacité de notre approche. Principalement programmé en OCaMl, cet outil utilise des librairies Java pour la construction des structures combinatoires, la librairie MuPAD-Combinat pour manipuler ces structures combinatoires, ainsi qu'un solveur de contraintes basé sur une technique de résolution randomisée et issu de l'outil GaTeL pour résoudre les prédicats.

En 1991, Pascale Thévenod-Fosse et Hélène Waeselynck du LAAS (Toulouse) ont défini pour la premiere fois le test statistique structurel. Cette méthode est basée sur la construction d'une distribution du domaine des entrées qui nécessite la résolution d'un système d'équations qui a autant d'équations que de chemins. Par conséquent, s'il y a un nombre infini ou trop important de chemins, cette construction doit être effectuée de manière empirique, ce qui freine son automatisation et donc sa généralisation, le tirage étant bien sûr automatisé.

Les distributions utilisées par notre méthode et celle du LAAS n'étant pas de même nature, nous ne pouvons comparer ces approches qu'expérimentalement. Nous avons donc repris plus de 10000 exécutions effectuées sur 2914 mutants, créés à partir de quatre fonctions issues d'un logiciel industriel et fournis par Thévenod-Fosse et Waeselynck. Ces expériences ont également servi:

- à évaluer la stabilité du pouvoir de détection des fautes de notre méthode: comme toute méthode basée sur un tirage aléatoire, il faut s'assurer que les résultats obtenus ne sont pas le fruit d'un tirage heureux mais qu'ils restent semblables d'une expérience à l'autre.

- estimer si un passage à l'échelle est possible: beaucoup de méthodes sont efficaces sur de petits programmes mais deviennent inopérantes dès que les programmes deviennent de taille réelle.

La première difficulté que nous avons rencontrée concerne l'environnement dans lequel nous avons exécuté ces expériences. Ainsi par exemple, l'ensemble des mutants à considérer avait peu de chance d'être exactement le même que celui de la méthode du LAAS car le comportement de certains d'entre eux dépendait de l'environnement dans lequel ils étaient exécutés.

Par ailleurs, la présence de variables globales non initialisées dans certaines fonctions a rendu l'exécution des tests très dépendante de l'environnement et en particulier de l'ordre dans lequel on exécute les tests. Ce qui a eu comme conséquence de rendre l'étude de la stabilité de notre approche plus délicate.

De plus, le nombre important de chemins tirables (environ 10^{30}) et de chemins infaisables (plus de 99%) pour une des fonctions a nécessité l'optimisation des structures combinatoires en fonction des propriétés structurelles du code afin de diminuer la proportion de chemins tirables (à environ 10^{20}) et la proportion de chemins infaisables (à 50%).

Malgré ces différentes difficultés, les résultats que nous avons obtenus sont comparables à ceux du LAAS et notre processus de génération de tests est complètement automatisé pour ces fonctions, contrairement à la méthode du LAAS qui a dû déterminer de manière empirique la distribution pour une de ces fonctions.

Conclusion Dans cette thèse, nous proposons une nouvelle méthode de test statistique basée sur l'utilisation des structures combinatoires et complètement automatisée. Une première application de notre approche au test statistique structurel a été étudiée et a permis de développer un prototype. Des expériences ont ainsi pu être réalisées afin d'étudier l'efficacité de notre approche en la comparant à une méthode de test statistique structurel existante. Les résultats obtenus sont comparables à ceux de cette méthode. De plus, notre approche s'avère stable et à l'avantage d'être complètement automatisée. Ces résultats expérimentaux obtenus pour des fonctions non triviales nous encouragent à continuer nos travaux dans cette voie et nous permettent également d'envisager un passage à l'échelle de notre approche et par conséquent le développement d'un outil destiné au monde industriel. De plus, ces travaux pourraient servir de base pour une nouvelle classe d'outils dans le domaine du test de logiciel.