Outil SANTE: Détection d'erreurs par analyse statique et test structurel des programmes C

Omar Chebaro

LIFC, Université de Franche-Comté, 25030 Besançon France
CEA, LIST, Laboratoire Sûreté des Logiciels, PC 94
91191 Gif-sur-Yvette France
omar.chebaro@cea.fr

Résumé. Ce papier présente un prototype appelé SANTE (Static ANalysis and TEsting), qui implémente une nouvelle méthode combinant l'analyse statique et le test structurel pour la détection des erreurs à l'exécution dans les programmes C. Tout d'abord, un outil d'analyse statique (Frama-C) est appelé pour générer des alarmes lorsqu'il ne peut pas garantir l'absence d'erreurs à l'exécution. Ensuite ces alarmes guident la génération de tests structurels dans PathCrawler qui va essayer de confirmer les alarmes en activant des bugs sur certains cas de test.

Mots clés: test structurel, analyse statique, erreurs à l'exécution, déboguage des programmes C, alarm-guided test generation.

1 Introduction

La validation des logiciels est une partie cruciale dans le cycle de leur développement. Deux techniques de vérification et de validation se sont démarquées au cours de ces dernières années : l'analyse statique et l'analyse dynamique. Ces deux techniques ont longtemps été considérées et étudiées comme deux domaines séparés.

Les points forts des deux techniques sont complémentaires. L'analyse statique (1) est imprécise mais correcte. En effet, ses résultats peuvent être plus faibles que souhaités, mais ils présentent l'avantage d'être valables pour l'ensemble des exécutions. L'analyse dynamique quant à elle est une méthode précise mais incomplète. En effet, pour un cas de test donné, l'analyse dynamique peut examiner le comportement exact du programme. Cependant cette analyse est incomplète en raison du grand nombre (voire infinité) de cas de tests possibles.

Pour résumer, si aucune menace d'erreur (d'un type particulier) n'est levée lors de l'analyse statique dans aucun des comportements possibles d'un programme donné alors il est certain que ce programme ne contient pas ce genre d'erreurs. Cependant, les erreurs potentielles signalées par cette analyse peuvent s'avérer de fausses alarmes. Par contre, même si l'analyse dynamique ne peut en général prouver l'absence d'erreurs dans un programme, une erreur détectée par cette analyse est forcément réelle.

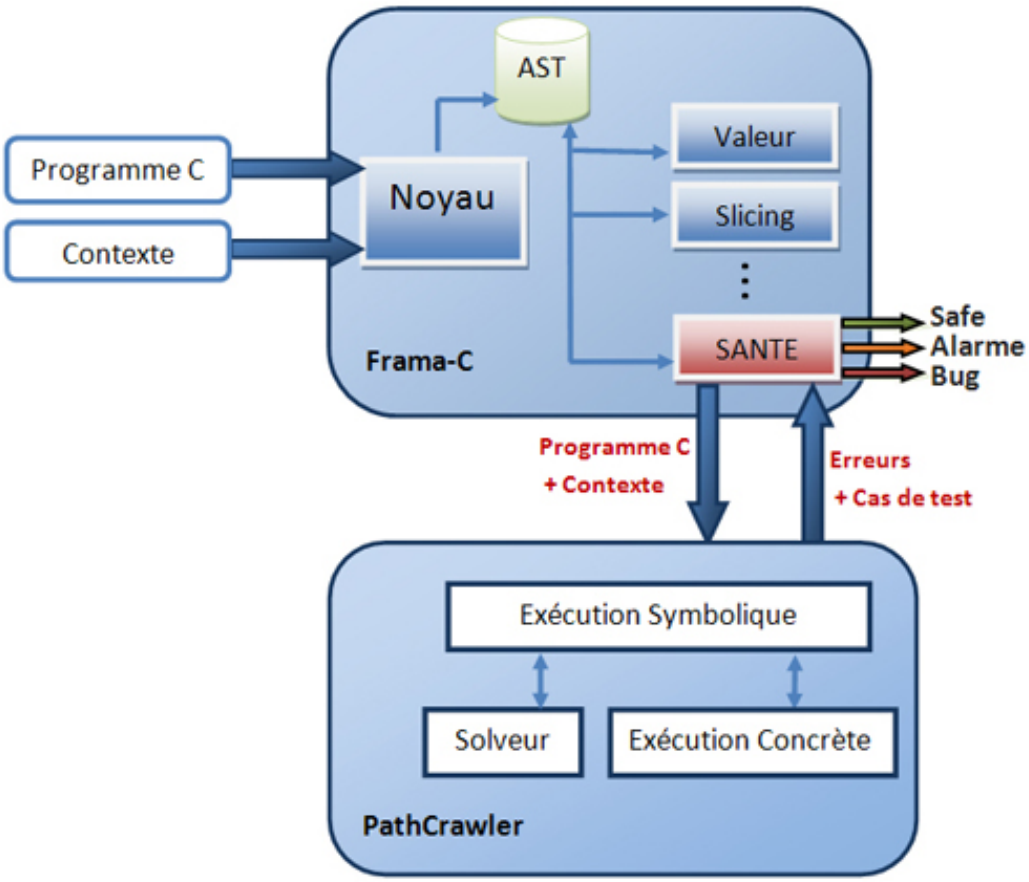

FIG. 1 – *SANTE : Architecture*

Ce papier décrit notre outil nommé SANTE (Static ANalysis and TEsting) qui implémente une méthode mixte de recherche des erreurs à l'exécution dans les programmes C par analyse statique et test structurel afin de rendre leur recherche plus précise, automatique et diminuer le nombre de fausses alertes. Le papier est organisé comme suit. La section 2 décrit notre outil et son implémentation. La section 3 donne quelques perspectives et conclut.

2 Implémentation

SANTE (2; 3) est basé sur deux outils : Frama-C, une plate-forme pour l'analyse statique des programmes C, et PathCrawler, un outil de génération de tests structurels.

Frama-C (4) est développé en collaboration entre le CEA LIST et le projet ProVal de l'INRIA Saclay. Il est organisé sous forme d'une architecture à greffons. Chaque greffon peut

faire un type d'analyse et se servir des résultats d'analyses faites par les autres greffons. Frama-C est distribué en open source sur (4) avec ses différents greffons.

Développé au CEA LIST, PathCrawler (5; 6) est un outil de génération de tests structurels pour les fonctions C. Il analyse le code source pour déduire les cas de test. Il cherche à satisfaire le critère de couverture de tous les chemins, ou le critère de couverture de tous les k-chemins (k-*path*) qui restreint la génération aux chemins avec au plus k itérations successives de chaque boucle. Il utilise la recherche en profondeur d'abord.

SANTE assemble ces deux outils hétérogènes utilisant différentes technologies (telles que l'interprétation abstraite et la programmation en logique avec contraintes). La figure 1 illustre l'architecture de SANTE. Notre choix d'implémentation a été de relier PathCrawler à Frama-C via un nouveau greffon dans Frama-C, et d'adapter PathCrawler afin qu'il accepte les informations fournies par d'autres greffons.

L'idée principale — détaillée dans (2) — est d'appeler l'analyse de valeurs (7) de Frama-C qui va calculer et sauvegarder les sur-ensembles des valeurs possibles des variables à chaque instruction du programme. Entre autres, ces ensembles peuvent être utilisés pour exclure la possibilité d'une erreur à l'exécution. L'analyse de valeurs va générer des alarmes pour les instructions pour lesquelles l'absence d'erreurs d'exécution n'est pas assurée par analyse statique. Par exemple, pour l'instruction $y = x/z$, le gréffon émet *"Alarme : z peut être 0 !"* et retourne la condition caractérisant un état d'erreur *(z == 0)* si 0 est contenu dans le sur-ensemble de valeurs calculées pour z avant cette instruction.

Ensuite, ces alarmes sont transférées à PathCrawler, qui les utilisera pour guider la génération de tests. PathCrawler essaie de confirmer les alarmes en activant des erreurs sur certains cas de test lors de son parcours de tous les chemins ou k-chemins. Nous appelons cette technique *"alarm-guided test generation"*. Cette approche passe par la génération automatique des branches d'erreurs correspondant à la transition symbolique à couvrir (l'instruction menaçante) avec des conditions sur les variables caractérisant un état d'erreur. Dans l'exemple précédant, si l'alarme est émise, l'instruction menaçante `y=x/z ;` sera remplacée par

```
if(z==0) storeBugAndExit(); else y=x/z;
```

Nous avons adapté l'algorithme de génération de PathCrawler pour qu'il cherche à couvrir ces branches obtenues à partir des résultats de l'analyse statique. En cas de réussite, PathCrawler signale une erreur à l'exécution et arrête l'exécution du cas de test en cours. Les techniques d'évaluation symbolique à l'aide de la programmation par contraintes (8) sont utilisées pour concrétiser la menace et générer un cas de test.

Notre prototype SANTE traite actuellement le langage C avec les tableaux, pointeurs, conditionnels, boucles et appels de fonctions. Il prend en entrée le code source du programme sous test, et le contexte de test qui comprend les domaines des variables d'entrée et éventuellement des préconditions. Il produit en sortie les erreurs trouvées avec des cas de tests activant ces erreurs, et les menaces restantes qui sont de fausses alarmes (faux positifs) ou des erreurs qu'on n'a pas pu confirmer à cause de l'incomplétude du test. Un travail en cours vise à enregistrer ces résultats dans l'AST de Frama-C, pour que d'autres greffons puissent les exploiter par la suite.

3 Conclusion

Les premiers résultats obtenus sont prometteurs (2). Ils montrent que l'outil SANTE combinant l'analyse statique et l'analyse dynamique est plus performant que chacun de ces outils utilisés séparément. Il est plus précis qu'un analyseur statique et plus efficace en termes de temps et de nombre d'erreurs détectées qu'un outil de test structurel.

Les perspectives de travail incluent la poursuite des recherches visant à éliminer les alarmes non confirmées, mieux traiter d'autres types d'alarmes (par exemple les pointeurs non valides, les variables non initialisées) et intégrer le slicing de programme qui pourrait simplifier le programme par rapport à la menace détectée et donc accélerer la phase de géneration de test.

Remerciements. Je remercie mes encadrants Alain Giorgetti, Jacques Julliand et Nikolai Kosmatov pour leurs nombreux conseils et discussions ainsi que tous les membres des équipes Frama-C et PathCrawler d'avoir fourni les outils et de leur disponibilité.

Références

[1] Nielson, F., Nielson, H.R., Hankin, C. : Principles of Program Analysis. Springer-Verlag New York, Inc., Secaucus, NJ, USA (1999)

[2] Chebaro, O., Kosmatov, N., Giorgetti, A., Julliand, J. : Combining static analysis and test generation for C program debugging. In Fraser, G., Gargantini, A., eds. : TAP'10, 4th Int. Conf. on Tests and Proofs. Volume 6143 of LNCS., Malaga, Spain (2010) (A paraître).

[3] Chebaro, O., Kosmatov, N., Giorgetti, A., Julliand, J. : Combining Frama-C and PathCrawler for C program debugging. extended abstract. In : GDR GPL 2010, 2èmes journées nationales du Groupement de recherche CNRS du Génie de la programmation et du logiciel, Pau, France (2010) 217–218

[4] Frama-C : Framework for static analysis of C programs (2007-2010) http ://www.frama-c.com/.

[5] Botella, B., Delahaye, M., Hong-Tuan-Ha, S., Kosmatov, N., Mouy, P., Roger, M., Williams, N. : Automating structural testing of C programs : Experience with PathCrawler. In : AST, Vancouver, Canada, IEEE Computer Society (2009) 70–78

[6] Williams, N., Marre, B., Mouy, P. : On-the-fly generation of structural tests for C functions. In : ICSSEA'03, Paris (October 2003)

[7] Canet, G., Cuoq, P., Monate, B. : A value analysis for C programs. IEEE International Workshop on Source Code Analysis and Manipulation (2009) 123–124

[8] Kosmatov, N. : Chapter XI : Constraint-Based Techniques for Software Testing. Advances in Intelligent Information Technologies Book Series. In : Artificial Intelligence Applications for Improved Software Engineering Development : New Prospects. IGI Global (2009) ISBN : 1605667587.

Omar Chebaro

Summary

This paper presents our tool prototype called SANTE (Static ANalysis and TEsting), implementing a new combination of static analysis and structural program testing for detection of run-time errors in C programs. First, a static analysis tool (Frama-C) is called to generate alarms when it cannot ensure the absence of run-time errors. Second, these alarms guide a structural test generation tool (PathCrawler) trying to confirm alarms by activating bugs on some test cases.