Sujet de Thèse
Model Checking et Test Logiciel

Sami Evangelista et Kaïs Klai

LIPN, CNRS UMR 7030
Université Paris 13
99 avenue Jean-Baptiste Clément
F-93430 Villetaneuse, France

{sami.evangelista,kais.klai}@lipn.univ-paris13.fr

1 Contexte scientifique et problématique

1.1 Vérification formelle

L'évolution des systèmes distribués se caractérise par une complexité croissante et un rôle toujours plus critique. La vérification de leurs propriétés est reconnue comme un problème difficile. Les logiciels orchestrant de tels systèmes doivent réagir correctement et en particulier face aux situations critiques. La vérification de la correction d'un système vis à vis d'une spécification peut se faire de deux manières : par le test ou par la vérification formelle.

D'un coté, la vérification par test est la technique la plus souvent utilisée dans le monde industriel. Elle consiste à exécuter le logiciel sur des données d'entrée et à évaluer chaque couple entrée-sortie produit par rapport aux spécifications du logiciel, en utilisant un oracle dédié[AO08]. Le test permet de révéler la présence de fautes qu'un système pourrait contenir, mais il n'a aucun moyen de prouver qu'un tel système est exempt de fautes. En effet, le test n'est pas exhaustif : il opère sur un sous ensemble d'exécutions possibles (une sous-approximation). Le fait que ce sous-ensemble vérifie les propriétés désirées n'implique pas forcément que le système les vérifie. Le deuxième obstacle est que les différents cas nécessaires pour tester un logiciel sont de plus en plus nombreux et difficiles à trouver. On parle de couverture structurelle/fonctionnelle : les cas de test doivent couvrir un maximum de scénarios possibles.

D'un autre coté, les méthodes formelles de vérification de systèmes ont pour objectif d'assurer la fiabilité des systèmes, c'est-à-dire leur bonne spécification et l'absence d'erreur. Idéalement, pour concevoir le logiciel d'un système concurrent donné, il faudrait spécifier formellement ce système à l'aide d'un modèle mathématique à partir duquel on pourrait raisonner et vérifier les propriétés attendues. En réalité, ce processus se heurte à plusieurs problèmes d'ordre pratique et théorique. Un premier obstacle apparait au niveau de la définition du langage de spécification utilisé. Si celui-ci est trop expressif, alors on ne peut pas, mathématiquement, l'analyser automatiquement. Un second obstacle à la vérification formelle des systèmes complexe, en particulier celles basées sur la technique du

model checking[CGP99], est l'explosion combinatoire des états possibles des systèmes. Malgré les avancées spectaculaires de la technologie des ordinateurs, il arrive que l'on soit incapable d'analyser intégralement des systèmes par manque d'espace mémoire ou de temps.

Par ailleurs, les logiciels critiques étant de plus en plus complexes, l'utilisation d'une seule technique de vérification peut se révéler insuffisante. Afin de guider le test à explorer des parties distinctes du modèle dans le but d'améliorer le taux de révélation des fautes, le model checker peut-être utilisé pour extraire des informations du modèle du systèmes [ER20] ou vérifier des hypothèses d'exploration pour les composants logiciels à l'aide de l'apprentissage automatique [GMN+20]. Comme le model checking permet de générer des contre exemples pour invalider des propriétés, cette aptitude s'est révélée exploitable pour générer des entrées de tests concrets [KEB+20,KEB+20]. Lors de la phase de test, la génération d'une référence (i. e. Oracle) est incontournable pour évaluer le résultat fourni par le système cible.

2 Objectifs de la thèse

Cette thèse s'intéresse à la combinaison du model checking et le test à bas de modèles. Nous avons deux objectifs qui permettrons des contributions complémentaires pour les deux communautés :

- Définir un cadre d'expression des critères de couvertures à l'aide de traces observables qui permettent d'améliorer le potentiel de révélation des fautes dans la phase de test. Après la formalisation des critères structurels élémentaires, nous visons, dans un premier temps, à étendre l'étude à des critères structurels plus élaborés puis à des critères comportementaux visant la synchronisation et la concurrence [OT19]. Par la suite, nous allons exploiter le potentiel des graphes d'observation symbolique pour définir un formalisme de sélection de traces concrètes à partir des traces symboliques permettant la génération d'entrées de tests. Ces propositions seront par la suite appliquées à l'automatisation des tests des solutions web en fournissant un générateur de drivers de tests
- Parallèlement à cet objectif de dédier le model checking au test, un autre objectif de cette thèse sera de repousser d'avantage les limites du problème de l'explosion combinatoire dont souffre les approches de vérification de modèles. A cet effet, nous envisageons d'étudier l'extension des algorithmes explicites de réduction d'ordre partiel [GW93] existants aux structures symboliques à base de diagrammes de décisions (p.e., BDD [Bry92]) et les intégrer aux abstractions de l'espaces d'états développées par l'équipe [HIK04,KP08b,KP08a,DLKPTM11]. Enfin, la parallélisation de ces algorithmes et une intégration dans l'outil de model checking parallèle `https://lipn.univ-paris13.fr/pmc-sog/` seront envisagées.

Références

AO08. Paul Ammann and Jeff Offutt. *Introduction to Software Testing*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

Bry92. Randal E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3) :293–318, 1992.

CGP99. E. M. Clarke, Orna Grumberg, and Doron Peled. *Model Checking*. MIT Press, 1999.

DLKPTM11. Alexandre Duret-Lutz, Kais Klai, Denis Poitrenaud, and Yann Thierry-Mieg. Self-loop aggregation product - a new hybrid approach to on-the-fly ltl model checking. In *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*, volume 6996 of *Lecture Notes in Computer Science*, pages 336–350, 2011.

ER20. Sajad Esfandyari and Vahid Rafe. Extracting combinatorial test parameters and their values using model checking and evolutionary algorithms. *Appl. Soft Comput.*, 91 :106219, 2020.

GMN+20. Khouloud Gaaloul, Claudio Menghi, Shiva Nejati, Lionel C. Briand, and David Wolfe. Mining assumptions for software components using machine learning. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 159–171, New York, NY, USA, 2020. Association for Computing Machinery.

GW93. Patrice Godefroid and Pierre Wolper. Partial-Order Methods for Temporal Verification. In *CONCUR'1993*, volume 715 of *lncs*, pages 233–246. springer, 1993.

HIK04. Serge Haddad, Jean-Michel Ilié, and Kais Klai. Design and evaluation of a symbolic and abstraction-based model checker. In *Automated Technology for Verification and Analysis : Second International Conference - ATVA*, volume 3299 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 2004.

KEB+20. Warda El Kholy, Mohamed El-Menshawy, Jamal Bentahar, Mounia Elqortobi, Amine Laarej, and Rachida Dssouli. Model checking intelligent avionics systems for test cases generation using multi-agent systems. *Expert Syst. Appl.*, 156 :113458, 2020.

KP08a. Kais Klai and Laure Petrucci. Modular construction of the symbolic observation graph. In *ACSD*, pages 88–97. IEEE, 2008.

KP08b. Kais Klai and Denis Poitrenaud. MC-SOG : An LTL model checker based on symbolic observation graphs. In *Petri Nets*, 2008.

OT19. Jeff Offutt and Sunitha Thummala. Testing concurrent user behavior of synchronous web applications with petri nets. *Software & Systems Modeling*, 18(2) :913–936, 2019.