Spécification légère et analyse de systèmes dynamiques munis de configurations riches

Nuno Macedo, Julien Brunel, David Chemouil, Alcino Cunha, Denis Kuperberg

Spécification légère et analyse de systèmes dynamiques munis de configurations riches*

Nuno Macedo
HASLab, INESC TEC & U. do Minho

Julien Brunel
ONERA/DTIS

David Chemouil
ONERA/DTIS

Alcino Cunha
HASLab, INESC TEC & U. do Minho

Denis Kuperberg
ENS Lyon

Résumé

Cet article constitue un bref résumé d'un article accepté et présenté à la conférence *Foundations of Software Engineering 2016,* intitulé *Lightweight Specification and Analysis of Dynamic Systems with Rich Configurations* [3].

Le *model-checking* est de plus en plus populaire dans les premières phases du processus de développement logiciel. Pour valider une conception logicielle, il faut généralement vérifier des propriétés structurelles et comportementales (ou temporelles). Malheureusement, la plupart des langages de spécification et des *model-checkers* excellent uniquement dans l'analyse de l'une ou l'autre sorte. Cela limite leur capacité à vérifier les systèmes dynamiques munis de configurations riches : des systèmes dont l'espace d'états est caractérisé par des propriétés structurelles, mais dont l'évolution devrait aussi vérifier des propriétés temporelles. Pour résoudre ce problème, nous proposons d'abord Electrum, une extension du langage de spécification Alloy avec des opérateurs de logique temporelle, où des configurations riches et des propriétés temporelles peuvent aisément être définies à la fois. Deux alternatives de techniques de *model-checking* sont ensuite proposées, l'une bornée et la seconde non-bornée, pour vérifier des systèmes exprimés dans ce langage, c'est-à-dire vérifier que les propriétés temporelles sont satisfaites dans toutes les configurations possibles.

1 Problématique

La spécification et la vérification des logiciels sont essentielles dans les premières phases de développement, car elles permettent au développeur de raisonner sur le système et ses propriétés et de détecter en temps opportun les erreurs de conception. Les cadres dits « légers », en ce sens qu'ils fournissent un langage formel simple mais expressif et flexible, permettent à l'utilisateur de spécifier différentes classes de systèmes et de propriétés à différents niveaux d'abstraction, et ils sont accompagnés d'outils qui automatisent leur analyse, offrant un retour rapide sur la correction de la spécification.

Deux classes de propriétés sont particulièrement importantes à considérer : des propriétés *structurales* (ou statiques), typiquement exprimées dans une variante de la logique du premier ordre, qui traitent de la bonne formation de l'état du système ; et des propriétés *comportementales* (ou dynamiques), typiquement exprimées dans une logique temporelle, qui traitent de l'évolution de l'état du système. Bien que pas nécessairement dans la même mesure, la plupart des systèmes réalistes nécessitent la spécification et l'analyse des propriétés des deux classes. L'analyse des algorithmes distribués est une classe paradigmatique dont les propriétés comportementales devraient être vérifiées pour des topologies de réseau arbitraires, dans une plage spécifiée par des propriétés structurelles particulières. Nous désignons ces composantes de l'état du système, qui sont initialement arbitraires, mais qui restent inchangées à mesure que le système évolue, sous le nom de *configurations*. Une autre classe pertinente est celle des

*Financements : *European Regional Development Fund* (ERDF) *via Operational Programme for Competitiveness and Internationalisation* (COMPETE 2020) ; Fundação para a Ciência e a Tecnologia (FCT) projet POCI-01-0145-FEDER-016826 ; projet DGA/ANR C*x* (ref. ANR-13-ASTR-0006) ; *fondation STAE* (IFSE, BRIefcaSE).

```
open util/ordering[Id]
sig Id {}
sig Process {
        id: one Id,
        succ: one Process,
        var toSend: set Id }
var sig elected in Process {}
pred step [p: Process] {
  some pid: p.toSend {
    p.toSend' = p.toSend - pid
    p.succ.toSend' = p.succ.toSend + (pid - prevs[p.succ.id]) }}
fact { always { all p: Process | step[p] or step [succ.p] or skip[p] }}
...
check { (some Process and Progress) => sometime some elected } for 4
```

FIGURE 1 – Extrait de code source Electrum pour l'exemple *leader election* [1]

lignes de produits logiciels. Ainsi, les systèmes dynamiques avec configurations riches sont au centre de ce travail, et concrètement, cette classe de systèmes présente les exigences suivantes :

R1 Une distinction claire entre la spécification de la configuration du système et l'évolution du système ;

R2 Des configurations contraintes par des propriétés structurelles riches (comme l'héritage, des relations complexes entre des entités ou des propriétés d'accessibilité) ;

R3 Une spécification déclarative de l'évolution du système (les actions possibles affectant l'état), éventuellement au moyen d'idiomes variés ;

R4 La nécessité de vérifier les propriétés (temporelles) de sûreté et de vivacité du système spécifié.

Malheureusement, la plupart des langages de spécification formels (et les *model-checkers* associés) ne sont pas conçus ni optimisés pour analyser des problèmes de ce type. Par exemple, la plupart des *model-checkers* standard ne fonctionnent bien qu'avec des configurations fixes, tandis que les langages plus orientés vers l'analyse des propriétés structurales, habituellement sans support natif pour la logique temporelle, exigent que l'utilisateur vérifie les propriétés comportementales par des mécanismes *ad hoc*.

2 Résultats

Notre travail vise précisément à combler ce créneau, et propose un langage et un *model-checker* adaptés à l'analyse légère de systèmes dynamiques avec de riches configurations. Concrètement, nous proposons : *(a)* Electrum, un langage de spécification formel (cf. figure 1), inspiré par Alloy [1] et TLA+ [2] (deux langages de spécification très utilisés de nos jours), qui simplifie la spécification de systèmes présentant toutes les exigences définies ci-dessus ; *(b)* Deux techniques de *model-checking*, l'une bornée et l'autre non bornée, pour vérifier les systèmes exprimés dans un tel langage, à savoir vérifier que chaque propriété temporelle souhaitable est valable pour toute configuration possible.

Comparativement à Alloy et TLC (le *model-checker* associé à TLA+), les performances de nos outils se situent dans le même ordre. Sur le plan de l'expressivité pour l'utilisateur, nous pensons qu'Electrum se situe sur un créneau intéressant pour la spécification de systèmes arborant des propriétés comportementales et des configurations riches.

Références

[1] D. Jackson. *Software Abstractions : Logic, Language, and Analysis*. The MIT Press, 2006.

[2] L. Lamport. *Specifying Systems, The* TLA$^+$ *Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[3] N. Macedo, J. Brunel, D. Chemouil, A. Cunha, and D. Kuperberg. Lightweight specification and analysis of dynamic systems with rich configurations. In *Proc. of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 373–383, New York, NY, USA, 2016. ACM.