Simulation symbolique et génération de tests pour les programmes synchrones ou hybrides

Timothy Bourke et Marc Pouzet

18 décembre 2017

L'équipe développe le langage synchrone Zélus [1] qui permet de combiner les constructions d'un langage synchrone (p. ex., typiquement pour programmer le logiciel de contrôle) et des équations différentielles ordinaires (ODEs) pour modéliser un environnement physique. Son compilateur produit du code séquentiel qui utilise un solveur numérique pour calculer une solution approchée des équations différentielles à des instants successifs. L'architecture du compilateur Zélus est suffisamment générique pour pouvoir utiliser n'importe quel solveur numérique d'ODEs. Nous avons récemment expérimenté une nouvelle manière de simuler un modèle, en remplaçant la simulation numérique par une simulation symbolique [1] [2] : plutôt que de calculer une solution approchée des équations, on calcule une enveloppe qui surapproxime l'ensemble des solutions possibles.

Nous aimerions approfondir cette approche pour mieux tester les modèles (trouver des bugs et augmenter la confiance que le modèle est juste). Il y a deux voies prometteuses :

a) Combiner la simulation symbolique et l'analyse statique, à partir des travaux d'Aditya Kanade et ses collègues [3]. On définira pour cela une extension à Zélus permettant de décrire des contraintes (dans le stage, on se limitera à des contraintes linéaires entre variables numériques et des contraintes booléennes), dans l'esprit du langage Lutin [4] en intégrant un solveur de contraintes de type SMT. L'objectif est de disposer d'une extension permettant à la fois de programmer un système avec son environnement physique et d'écrire des oracles pour le tester.

b) Une voie autre consiste à adapter les idées et principes du Property Based Testing introduits par Claessen et Hughes dans l'outil QuickCheck [2] [5], au langage Zélus. En effet, QuickCheck offre un moyen puissant de faire du test aléatoire au point que ses principes sont maintenant intégrés dans la plupart des langages de programmation. L'intégration de ces principes et sa mise en œuvre dans un langage synchrone ou de modélisation de systèmes hybrides est complètement ouverte.

   Ce travail s'effectuera dans le cadre de notre collaboration avec Koen Claessen et John Hugues (Chalmers Univ., Suède).

1. http://zelus.di.ens.fr
2. https://hal.inria.fr/hal-01575621v2/document
3. outil http://www.iisc-seal.net/slsf
4. http://www-verimag.imag.fr/Lutin.html?lang=fr
5. http://www.cs.tufts.edu/~nr/cs257/archive/john-hughes/quick.pdf

Ce stage s'adresse à un étudiant ayant un fort goût pour les langages, les méthodes formelles, le test et la compilation.

Références

[1] G. Baudart, T. Bourke, and M. Pouzet. Symbolic simulation of dataflow synchronous programs with timers. In *Proceedings of the 12th Forum on Specification and Design Languages (FDL 2017)*, page TBA, Verona, Italy, Sept. 2017.

[2] K. Claessen and J. Hughes. QuickCheck : A lightweight tool for random testing of haskell programs. In *Proceedings of the 5th ACM SIGPLAN International Conference on Functional Programming (ICFP 2000)*, volume 35 of *ACM SIGPLAN Notices*, pages 268–279, Montreal, Canada, Sept. 2000. ACM, ACM Press.