Stage de recherche en informatique: Génération de code pour le contrôle des systèmes synchrones critique

Sid Touati*
Frédéric Mallet†

2021

Lieu

Laboratoire I3S et INRIA Sophia Antipolis, France. Equipe-projet KAIROS.

1 Motivations

Les méthodes formelles de modélisation des systèmes critiques proposent des langages précis de description de comportement, dont la sémantique est plus riche que celle des langages de programmation impérative. Grâce à ces méthodes formelles, il est possible de garantir ou vérifier certaines propriétés exigées pour le bon fonctionnement d'une application. Ces propriétés peuvent être de différentes natures: performances (temps d'exécution, consommation mémoire), de correction (une fonction calcule-t-elle réellement ce qu'elle est sensée calculer ?), de comportement (l'application réagit-elle à son environnement au bon rythme ?). Il existe toute une communauté de recherche académique et industrielle autour des méthodes formelles de description d'application critiques (aviation, transport, armement, industrie spatiale), qui travaille depuis plusieurs années autour des langages et des outils de d'analyse et de manipulation des modèles.

Cependant, ces langages formels ont encore du mal à être acceptés par une majorité de concepteurs ou de programmeurs. Deux explications peuvent être apportées :

1. Mentalité algorithmique des programmeurs : les langages formels proposent de modéliser des applications avec une vision assez éloignée de

*Professeur à l'université Côte d'Azur. Sid.Touati@inria.fr
†Professeur à l'université Côte d'Azur. Frederic.Mallet@inria.fr

l'algorithmique, l'application n'est pas décrite avec son comportement dans un langage de programmation classique. Cela exige de raisonner avec une vue d'esprit différente de celle de l'informaticien, dans un niveau d'abstraction plus élevé. Il existe plusieurs sortes de modélisations haut niveau : graphiques (UML), fonctionnelles (équations mathématiques), synchrones (description minutieuse et formelle des horloges qui rythment le système), etc.

2. Absence d'outils fiables de génération de code efficace : avec les langages formels, il est possible d'analyser le système, de le manipuler, mais il n'est pas encore possible de générer du code efficace, comme le feraient des compilateurs. Par code efficace, nous entendons un programme source puis un code binaire qui s'exécute avec des performances satisfaisantes sur des processeurs physiques. Actuellement, il existe des outils de simulation de modèles formels hélas peu performants.

Afin de motiver les concepteurs d'utiliser des méthodes formels, nous orientons nos efforts de recherche durant les prochaines années à résoudre le 2e point ci-dessus. Nous souhaitons réfléchir et implémenter des méthodes automatiques de génération de code efficace à partir d'une description haut niveau, de type CCSL (*Logical Clock Calculus Algebra*). Le code généré se fera dans un langage impératif connu comme le C et le C++, qui sera ensuite destiné à une optimisation avancée via un compilateur. Le code généré devra être *compiler friendly*, à savoir être assez clair pour le compilateur et bien disposé à des techniques d'optimisation de code bas niveau. Le code optimisé est prévu pour être exécuté sur un système embarqué ou cyber-physique avec contraintes fortes sur les performances (temps d'exécution ou consommation mémoire).

Présentation du sujet de stage de recherche

Ce stage a pour objectif de générer du code C ou C++ à partir d'une description CCSL (*Logical Clock Calculus Algebra*). CCSL permet de décrire non pas une application entièrement, mais uniquement le comportement de ses horloges. Une horloge permet de déclencher ou pas l'exécution d'une fonction ou d'une tache à un instant précis. CCSL n'est pas un langage de programmation qui permet d'exprimer l'algorithmique des fonctions, il exprime uniquement l'instant de leur exécution. Le code généré doit pouvoir permettre de calculer les valeurs de ces horloges logiques. Il y aura trois aspects principaux à aborder:

1. Au niveau de la spécification formelle avec CCSL: le formalisme de CCSL permet une grande liberté de description des horloges avec un algèbre propre. Suite à cela, un outil d'analyse existant permet de

déduire l'espace des valeurs possibles des horloges et de le modéliser avec avec un automate.

2. Une fois l'automate construit, l'étudiant devra réfléchir aux méthodes de génération automatique de code C ou C++. Dans la littérature, il y a plusieurs schémas de génération de code pour automates, qui ne sont pas conçus dans une optique de performances, mais plutôt dans une optique de correction sémantique.

3. Implémenter d'un générateur de code à partir d'une description CCSL.

Ce sujet de stage ouvre à une poursuite en thèse de doctorat.

Prérequis souhaités

Compilation, automates, algèbre.

Références

CCSL2009 Charles André. *Syntax and Semantics of the Clock Constraint Specification Language (CCSL).* RR-6925, INRIA (2009). https://hal.inria.fr/inria-00384077/

CCSL2008 Frédéric Mallet, Charles André and Robert de Simone. *CCSL: specifying clock constraints with UML/Marte.* Innovations in Systems and Software Engineering 4(3): 309-314, Springer, 2008. https://hal.inria.fr/inria-00371371/