Nouvelles fonctionnalités pour un compilateur Lustre formellement vérifié avec Coq

Timothy.Bourke@inria.fr and Marc.Pouzet@ens.fr

Printemps 2020 (mi-mars/fin août)
Équipe PARKAS, DI, Ecole normale supérieure
45 rue d'Ulm, 75 230 Paris
http://parkas.di.ens.fr

Ce stage contribuera au développement de Vélus [1], un compilateur spécifié et vérifié avec Coq pour le langage synchrone Lustre. Ce compilateur est en cours de développement dans l'équipe Inria PARKAS depuis quelques années. Par le passé, nous avons réussi à intégrer le compilateur CompCert [10] [2], et à démontrer la correction de la compilation d'un sous-ensemble de Lustre vers du code assembleur [JFLA17, PLDI17], traiter les arguments cadencés [JFLA19] et, très récemment, étendre la modèle sémantique et les passes de compilation pour permettre la réinitialisation modulaire [POPL20].

Cette année, il y a plusieurs possibilités de stage autour du compilateur Vélus. Les quatres sujets suivants peuvent vous donner une idée de ce qui est possible, mais passez nous voir pour en discuter en détail !

Analyse d'initialisation et retard non initialisé dans Vélus L'opérateur « pre » de Lustre a une valeur indéfinie au premier instant. Il est indispensable de montrer que la sémantique d'un programme ne dépend pas de cette valeur. Une analyse d'initialisation par typage est définie dans l'article [8] et elle est implémentée dans le compilateur industriel KCG de SCADE 6. L'objectif de ce stage est de la formaliser, de la mettre en œuvre dans le compilateur Vélus (celui-ci n'a que « fby » pour l'instant) et d'en prouver la correction complète, c'est-à-dire jusqu'au code exécutable produit par CompCert.

Justification de l'existence d'un modèle sémantique pour un programme Lustre bien formé Le théorème de correction de Vélus établit un lien formel entre la sémantique flot de données de Lustre et la sémantique impérative du code assembleur généré par la chaîne de compilation. Une faiblesse du théorème actuel est qu'il suppose que une sémantique flot de données existe pour tous les programmes acceptés par le compilateur. Or, ce fait mérite, lui aussi, d'être démontré. D'abord, pour assurer que les analyses statiques censées rejeter les programmes mal formés, tel qu'il existe pour le typage, les horloges [7] et la causalité, sont correctes. Mais aussi pour imposer une obligation de preuve aux utilisateurs qui devraient assurer qu'un programme ne fait

1. https://velus.inria.fr
2. http://compcert.inria.fr

jamais un calcul « non défini » comme, par exemple, une division entier par zéro. Nous proposons de développer une sémantique où la possibilité d'une erreur est représentée explicitement [2, 3] et de l'utiliser pour étendre et compléter une preuve existante basée sur un interprète de Lustre normalisé.

Une sémantique exécutable fonctionnelle d'un langage synchrone avec automates Le langage synchrone Scade 6 [6] au coeur de l'outil de développement SCADE [3] a été conçu sur le principe suivant : (1) la définition d'un language noyau data-flow synchrone dont les sémantiques statiques et dynamiques sont définies rigoureusements. Il sert de *langage de bas niveau typé* duquel le code séquentiel est produit. C'est, pour l'essentiel, le langage que compile Velus. (2) un langage complet bâti au dessus, comprenant en particulier des structures de programmation plus riches tels que les automates hiérarchiques, et une traduction source-à-source vers le langage noyau.

L'objectif du stage est d'étudier une sémantique directe du langage source, avant compilation, c'est-à-dire avant la normalisation, le scheduling et sans traduction vers le langage noyau. On considèrera un langage combinant data-flow et des structures de contrôle tels que les automates hierarchiques. On souhaite, en particulier, disposer d'une sémantique exécutable, autrement dit, d'un interprète. C'est utile pour comparer les résultats avec le code généré et présente un second avantage, moins immédiat mais essentiel : permettre de simuler, au moment de la conception, des modèles partiels, éventuellement pas entièrement terminés, sémantiquement corrects mais rejetés par le calcul d'horloge ou l'analyse de causalité parce que celles-ci ne sont pas suffisamment expressives. Cette sémantique fonctionnelle pourrait être écrite en Coq pour servir de base à des preuves de correction futures et disposer d'un interprète en Ocaml. Mais ce n'est pas obligatoire ; on peut dans un premier temps écrire un interprète directement en OCaml.

Ce travail s'appuiera sur plusieurs travaux : l'article de Caspi et Pouzet [3] sur une sémantique fonctionnelle par co-itération (dont les notes de cours de cette année sont adaptées) ; une sémantique relationnelle (non exécutable) pour le noyau data-flow et automates de Scade [4] ; l'article de Hamon [5] qui définit une sémantique dénotationnelle de StateFlow à partir de laquelle il présente un interprète et une méthode de compilation.

Représenter les dépendances causales par des types L'analyse des causalités entre signaux est une des questions les plus étudiées dans les langages de description de systèmes réactifs et/ou concurrents. Le modèle synchrone, par l'introduction d'une échelle de temps global, simplifie cette analyse en l'a ramenant à une question plus simple. On se limite au calcul de dépendances instantanées seulement.

Ce problème plus simple peut-être simplifié encore en interdisant les dépendances conditionnelles — c'est le cas de Lustre, Lucid Synchrone et Zelus — ou en acceptant certaines formes, comme c'est le cas avec les langages Esterel et Signal. Mais la encore, on se limite à de la propagation de faits [1, 11].

Pour ce stage, on propose d'étudier la causalité d'un langage de programmation data-flow synchrone *fonctionnel*, dans lequel il est possible d'écrire des

3. https://www.ansys.com/fr-fr/products/embedded-software/ansys-scade-suite

fonction de suites et des fonctions d'ordre supérieur, en particulier des fonctions ayant des fonction de suites en paramètre et/ou en résultat. L'analyse de causalité sans dépendances conditionnelles peut alors s'exprimer comme un problème de typage avec une relation de sous-typage en exploitant les algorithmes connus [12]. La propriété attendue pour les programmes qui ont passé l'analyse de causalité est double : 1/ les programmes acceptés produisent une valeur à chaque instant (réactivité) ; 2/ ils peuvent être compilés vers du code séquentiel. La preuve de correction du système a été établie (sur papier) pour le cas 1er ordre mais pas encore pour l'ordre supérieur. L'objectif du stage est de faire cette preuve, ce qui constituerait une première. Si le temps le permet, c'est l'occasion de voir si et comment exprimer certaines formes de dépendances conditionnelles que l'on a en Esterel mais qui sont rejetées en Lustre (et Zelus). Puis dans étudier les conséquences en terme de génération de code.

Ce travail s'appuiera sur l'article de Cuoq et al. [9] qui exprime l'analyse de causalité par typage, pour la première fois, et l'article de Benveniste et al. [HSCC14] qui décrit une analyse de causalité par typage un peu différente, appliquée au langage Zelus, et établit la correction de l'analyse dans le cas d'un langage premier ordre. Les expériences pourront être menées en utilisant/modifiant le compilateur Zelus.

Prerequisites Le stage s'adresse à un étudiant ayant un goût prononcé pour les langages de programmation, leur conception, leur sémantique et leur compilation. Les développements logiciels seront réalisés en OCaml ou en Coq.

La volonté à discuter et à collaborer de manière constructive avec d'autres chercheurs.

- Une formation solide dans la conception de langages de programmation (*réquis*)
- Expérience de la programmation fonctionnelle. (*souhaité*)
- Expérience de la modélisation et vérification formelle avec un assistant de preuve (Coq, HOL, PVS, Isabelle, Lean, etcétéra). (*souhaité*)
- Expérience préalable de la programmation synchrone. (*optional*)

Informations administratives Ce stage aura lieu dans l'équipe Inria PARKAS au DI de l'ENS Paris : http://www.di.ens.fr/ParkasTeam.html

Directeurs. Timothy Bourke et Marc Pouzet.

Le stagiaire recevra une allocation en fonction de son statut administratif. Ce stage est financé par le projet ANR *FidelR*.

Programme doctoral. Les sujets décrits ici peuvent se prolonger par une thèse. L'équipe PARKAS a de solides relations de travail avec l'équipe de développement du compilateur Scade chez Ansys et des équipes de recherche et développement chez SAFRAN et Airbus.

Références

[HSCC14] Albert Benveniste et al. "A Type-Based Analysis of Causality Loops in Hybrid Modelers". In : *Proc. 17th Int. Conf. on Hybrid Systems : Computation and Control (HSCC 2014)*. Sous la dir. de Martin Fränzle et John Lygeros. Berlin, Germany : ACM Press, avr. 2014, p. 71-82. DOI : 10.1145/2562059.2562125. URL : http://www.tbrk.org/papers/abstracts.html#hscc14.

[1] Gérard Berry. *The Constructive Semantics of Pure Esterel*. Draft Version 3. Sophia-Antipolis, France, déc. 2002. URL : http://www-sop.inria.fr/members/Gerard.Berry/Papers/EsterelConstructiveBook.pdf.

[2] Sylvain Boulmé et Grégoire Hamon. "Certifying Synchrony for Free". In : *Proc. 8th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR 2001)*. Sous la dir. de Robert Nieuwenhuis et Andrei Voronkov. T. 2250. LNCS. Havana, Cuba : Springer, déc. 2001, p. 495-506. URL : https://hal.archives-ouvertes.fr/hal-01571762.

[POPL20] Timothy Bourke, Lélio Brun et Marc Pouzet. "Mechanized Semantics and Verified Compilation for a Dataflow Synchronous Language with Reset". In : *Proc. 47th ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages (POPL 2020)*. New Orleans, LA, USA : ACM Press, jan. 2020, article 44. DOI : 10.1145/3371112. URL : http://www.tbrk.org/papers/abstracts.html#popl2020.

[JFLA19] Timothy Bourke et Marc Pouzet. "Arguments cadencés dans un compilateur Lustre vérifié". In : *30$^{\text{ièmes}}$ Journées Francophones des Langages Applicatifs (JFLA 2019)*. Sous la dir. de Nicolas Magaud et Zaynah Dargaye. Les Rousses, Haut-Jura, France, jan. 2019, p. 109-124. URL : https://hal.inria.fr/hal-02005639/document.

[PLDI17] Timothy Bourke et al. "A Formally Verified Compiler for Lustre". In : *Proc. 2017 ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI)*. Barcelona, Spain : ACM Press, juin 2017, p. 586-601. DOI : 10.1145/3062341.3062358. URL : http://www.tbrk.org/papers/abstracts.html#pldi2017.

[JFLA17] Timothy Bourke et al. "Vérification de la génération modulaire du code impératif pour Lustre". In : *28$^{\text{ièmes}}$ Journées Francophones des Langages Applicatifs (JFLA 2017)*. Sous la dir. de Julien Signoles et Sylvie Boldo. Gourette, Pyrénées, France, jan. 2017, p. 165-179. URL : http://www.tbrk.org/papers/abstracts.html#jfla17.

[3] Paul Caspi et Marc Pouzet. "A Co-iterative Characterization of Synchronous Stream Functions". In : *First Workshop on Coalgebraic Methods in Computer Science (CMCS'98)*. T. 11. ENTCS. Lisbon, Portugal : Elsevier Science, mar. 1998, p. 1-21. DOI : 10.1016/S1571-0661(04)00050-7.

[4] Jean-Louis Colaço, Grégoire Hamon et Marc Pouzet. "Mixing Signals and Modes in Synchronous Data-flow Systems". In : *Proc. 6th ACM Int. Conf. on Embedded Software (EMSOFT 2006)*. Sous la dir. de Sang Lyul Min et Yi Wang. Seoul, South Korea : ACM Press, oct. 2006, p. 73-82. URL : https://www.di.ens.fr/~pouzet/bib/emsoft06.pdf.

[5] Jean-Louis Colaço, Bruno Pagano et Marc Pouzet. "A Conservative Extension of Synchronous Data-flow with State Machines". In : *Proc. 5th ACM Int. Conf. on Embedded Software (EMSOFT 2005)*. Sous la dir. de Wayne Wolf. Jersey City, USA : ACM Press, sept. 2005, p. 173-182. DOI : 10.1145/1086228.1086261. URL : https://www.di.ens.fr/~pouzet/bib/emsoft05b.pdf.

[6] Jean-Louis Colaço, Bruno Pagano et Marc Pouzet. "Scade 6 : A Formal Language for Embedded Critical Software Development". In : *Proc. 11th Int. Symp. Theoretical Aspects of Software Engineering (TASE 2017)*. Nice, France : IEEE Computer Society, sept. 2017, p. 4-15. URL : https://hal.inria.fr/hal-01666470/document.

[7] Jean-Louis Colaço et Marc Pouzet. "Clocks as First Class Abstract Types". In : *Proc. 3rd Int. Conf. on Embedded Software (EMSOFT 2003)*. Sous la dir. de R. Alur et I. Lee. T. 2855. LNCS. Philadelphia, PA, USA : Springer, oct. 2003, p. 134-155. DOI : 10.1007/978-3-540-45212-6_10.

[8] Jean-Louis Colaço et Marc Pouzet. "Type-based initialization analysis of a synchronous dataflow language". In : *Int. J. Software Tools for Technology Transfer* 6.3 (août 2004), p. 245-255. URL : https://www.di.ens.fr/~pouzet/bib/sttt04.pdf.

[9] Pascal Cuoq et Marc Pouzet. "Modular Causality in a Synchronous Stream Language". In : *10th European Symposium on Programming (ESOP 2001), part of European Joint Conferences on Theory and Practice of Software (ETAPS 2001)*. Sous la dir. de David Sands. T. 2028. LNCS. Genova, Italy : Springer, avr. 2001, p. 237-251. DOI : 10.1007/3-540-45309-1_16.

[10] Xavier Leroy. "Formal verification of a realistic compiler". In : *Comms. ACM* 52.7 (2009), p. 107-115. URL : https://hal.inria.fr/inria-00415861/document.

[11] Michael Mendler, Thomas R. Shiple et Gérard Berry. "Constructive Boolean circuits and the exactness of timed ternary simulation". In : *Formal Methods in System Design* 40.3 (juin 2012), p. 283-329. DOI : 10.1007/s10703-012-0144-6.

[12] François Pottier. "Simplifying Subtyping Constraints : A Theory". In : *Information and Computation* 170.2 (nov. 2001), p. 153-183. DOI : 10.1006/inco.2001.2963.