Lucy-n : une extension n-synchrone de Lustre*

Louis Mandel Florence Plateau Marc Pouzet

LRI, Université Paris-Sud 11
INRIA Saclay
{mandel,plateau,pouzet}@lri.fr

Nous nous intéressons aux modèles et aux langages pour la programmation d'applications de traitement de flux ayant des contraintes de temps-réel comme les applications multimédias. Dans le cas d'une application vidéo par exemple, il faut traiter des flots infinis de pixels sur lesquels sont appliquées des opérations successives, ou *filtres*. Il y a des contraintes de temps-réel car la fréquence d'affichage des pixels ne peut être diminuée. Il y a également des contraintes de performance car des milliards d'opérations doivent être effectuées par seconde. Enfin, il y a des contraintes de sûreté : on veut garantir que le comportement de l'application est déterministe, sans blocage et que la mémoire nécessaire à l'exécution est bornée.

Afin de respecter ces contraintes, les ingénieurs du domaine sont habitués à modéliser leurs systèmes par des réseaux de processus de Kahn [7]. Dans ce modèle, des nœuds de calcul s'exécutent de manière concurrente et communiquent par des canaux munis de *mémoires tampon* (ou *buffers*) infinies de type *First In First Out*. La lecture est bloquante si le buffer est vide, et comme les buffers sont de taille infinie, l'écriture est non bloquante. On ne peut pas tester l'absence de valeur dans un buffer.

L'intérêt ce modèle de programmation est d'être concurrent tout en préservant le déterminisme. De plus, il peut être vu comme un modèle mathématique où chaque nœud correspond à une fonction sur les suites et le réseau correspond à la composition de ces fonctions, qui est aussi une fonction de suites. On sait ainsi raisonner de manière formelle sur les systèmes décrits dans ce formalisme. Néanmoins, l'inconvénient de ce modèle est qu'il ne donne pas de garanties sur l'exécution en temps-réel, l'absence de blocage et le caractère borné de la mémoire nécessaire à la communication.

Il est possible d'implémenter des réseaux de Kahn sans prendre le risque d'écritures dans des buffers pleins en mettant en place un mécanisme de rétroaction (ou *back pressure*). Ce mécanisme rend l'écriture bloquante. Ainsi, il permet d'éviter les débordements de capacité des buffers, mais il crée des blocages artificiels dans le cas où les tailles des buffers ont été sous-estimées. Ce problème peut être traité en augmentant les tailles durant l'exécution en cas de blocage dû à un manque de place [12]. Cependant, on aimerait avoir des garanties statiques sur l'exécution. Pour cela, il faut savoir calculer des tailles suffisantes pour les buffers si elles existent, et rejeter les réseaux qui nécessitent des buffers de taille infinie.

Le problème de savoir si un réseau de Kahn quelconque peut être exécuté en mémoire bornée est indécidable [2]. Pour avoir des garanties sur le caractère borné des buffers nécessaires, il faut donc se placer dans un sous-ensemble des réseaux de Kahn sur lesquels des informations supplémentaires sont disponibles. C'est l'approche choisie par le modèle des *Synchronous Dataflow* [8, 11] qui se restreignent à des réseaux de Kahn dans lesquels le nombre de valeurs produites et consommées par chaque nœud à chaque activation est connu statiquement. Cela permet de calculer un ordonnancement statique périodique et une taille suffisante pour les buffers pour cet ordonnancement.

L'approche consistant à se placer dans un sous-ensemble des réseaux de Kahn est aussi l'approche suivie par le modèle de programmation synchrone [1]. Dans ce modèle, on ne considère que les réseaux qui peuvent être exécutés sans buffers. Pour cela, chaque canal est associé à une *horloge* qui définit les instants où une valeur est présente sur le canal.

L'intérêt des langages synchrones est de permettre de spécifier formellement une application temps-réel tout en étant compilables vers du code exécutable. Ils expriment la concurrence des différents traite-

*Nous présentons ici un résumé étendu de notre article publié aux vingt et unièmes Journées Francophones des Langages Applicatifs [10].

ments, et sont donc bien adaptés à une compilation permettant d'exécuter parallèlement des filtres. Ils offrent des garanties fortes, comme le déterminisme, l'absence de blocage et un besoin en mémoire borné. La contrepartie des garanties fournies malgré une grande expressivité des rythmes est un manque de souplesse dans la composition des flots. La composition sans buffer n'est acceptée que si le calcul d'horloges sait vérifier l'égalité des horloges des flots que l'on compose.

La vérification des horloges en Lustre [3] et Lucid Synchrone [13] est un problème de typage et peut être décrite par un système de types [4, 6]. Chaque expression est associée à un type d'horloges et le compilateur vérifie que le programme respecte des contraintes sur ces types. Considérons par exemple la règle suivante :

$$\frac{H \vdash e_1 : ck \quad H \vdash e_2 : ck}{H \vdash e_1 + e_2 : ck}$$

Elle indique qu'une addition de deux flots e_1 et e_2 est correcte à condition que e_1 et e_2 aient le même type d'horloges ck. Dans ce cas, le type d'horloges du résultat $e_1 + e_2$ est aussi ck. Ainsi, pour typer un programme, les questions qui se posent concernent toutes l'égalité de types. L'idée du modèle n-synchrone [5] est d'assouplir cette contrainte d'égalité afin de permettre la composition de flots non nécessairement synchrones dès lors que leurs horloges sont proches l'une de l'autre. Si par insertion d'un buffer borné, un flot x dont l'horloge est de type ck peut être consommé un peu plus tard sur une horloge de type ck', on dira que ck est un sous-type de ck'. Cette relation sera notée $ck \ \mathbf{<:} \ ck'$. On étend alors le langage avec une construction `buffer` qui indique les points où il faut appliquer la règle de sous-typage :

$$\frac{H \vdash e : ck \quad ck \ \mathbf{<:} \ ck'}{H \vdash \texttt{buffer}\ e : ck'}$$

Le langage Lucy-n que nous proposons, est un langage synchrone flot de données semblable à Lustre, étendu avec l'opérateur de bufferisation. Par ailleurs, les expressions d'horloges (ce) peuvent être définies avec tout langage d'horloges s'évaluant vers des séquences booléennes infinies et muni des opérations suivantes :

- un test d'égalité des horloges, pour vérifier que les communications réalisées sans buffer sont synchrones ;
- un test d'*adaptabilité* des horloges, pour vérifier que les communications réalisées à travers un buffer sont n-synchrones ;
- une opération permettant de calculer une taille suffisante pour chaque buffer.

En Lustre, les expressions d'horloges peuvent être définies par n'importe quelle expression booléenne du langage. Ces expressions pouvant être arbitrairement compliquées, le test d'égalité des horloges se limite à de l'égalité syntaxique. Dans ce cadre, on peut difficilement imaginer un test d'adaptabilité et un calcul des tailles de buffers. On doit donc utiliser un langage d'horloges plus simple pour pouvoir utiliser des communications par buffers.

Dans l'article [10], nous présentons Lucy-n en limitant les expressions d'horloges à des mots binaires ultimement périodiques. Nous définissons ensuite le système de types qui vérifie que les types d'horloges sont égaux lorsque les communications se font sans buffers et collecte des contraintes de sous-typage lorsque les communications se font avec des buffers. Puis, nous montrons comment résoudre les contraintes de sous-typage en utilisant les techniques d'abstraction présentées dans [9]. Enfin, nous terminons l'article par un exemple d'application vidéo programmé en Lucy-n.

Références

[1] A. Benveniste, P. Caspi, S.A. Edwards, N. Halbwachs, P. Le Guernic, and R. de Simone. The synchronous languages 12 years later. *Proceedings of the IEEE*, 91(1), January 2003.

[2] J. Buck. *Scheduling Dynamic Dataflow Graphs with Bounded Memory Using the Token Flow Model*. PhD thesis, EECS Department, University of California, Berkeley, 1993.

[3] P. Caspi, D. Pilaud, N. Halbwachs, and J. A. Plaice. Lustre: a declarative language for real-time programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 178–188. ACM, 1987.

[4] Paul Caspi and Marc Pouzet. Synchronous Kahn Networks. In *ACM SIGPLAN International Conference on Functional Programming*, May 1996.

[5] A. Cohen, M. Duranton, C. Eisenbeis, C. Pagetti, F. Plateau, and M. Pouzet. N-Synchronous Kahn Networks: a Relaxed Model of Synchrony for Real-Time Systems. In *ACM International Conference on Principles of Programming Languages*, January 2006.

[6] Jean-Louis Colaço and Marc Pouzet. Clocks as First Class Abstract Types. In *Third International Conference on Embedded Software (EMSOFT'03)*, October 2003.

[7] G. Kahn. The semantics of a simple language for parallel programming. In J. L. Rosenfeld, editor, *Information processing*, pages 471–475, Stockholm, Sweden, August 1974.

[8] E. Lee and D. Messerschmitt. Synchronous dataflow. *IEEE Trans. Comput.*, 75(9), 1987.

[9] L. Mandel and F. Plateau. Abstraction d'horloges dans les systèmes synchrones flot de données. In *Vingtièmes Journées Francophones des Langages Applicatifs*, February 2009.

[10] L. Mandel, F. Plateau, and M. Pouzet. Lucy-n : une extension n-synchrone de Lustre. In *Vingt-et-unièmes Journées Francophones des Langages Applicatifs*, January 2010.

[11] T. M. Parks, J. L. Pino, and E. A. Lee. A comparison of synchronous and cycle-static dataflow. In *Proceedings of the 29th Asilomar Conference on Signals, Systems and Computers*, page 204, Washington, DC, USA, 1995. IEEE Computer Society.

[12] Thomas Martyn Parks. *Bounded scheduling of process networks.* PhD thesis, EECS Department, University of California, Berkeley, Berkeley, CA, USA, 1995.

[13] Marc Pouzet. *Lucid Synchrone, version 3. Tutorial and reference manual.* Université Paris-Sud, LRI, April 2006. Distribution available at: `www.lri.fr/~pouzet/lucid-synchrone`.