Thèmes et projets de recherche

Laure Gonnord

8 décembre 2008

1 Activités de recherche

1.1 Contexte général

Mes travaux de recherchent se placent dans le domaine de la vérification automatique de programmes, dont j'aborde ici les enjeux. Ensuite, je parlerai plus précisément des sujets que j'ai étudiés.

Le contexte général de mes travaux est celui de la spécification et de la vérification formelle de systèmes *embarqués*. Parmi ceux-ci, je m'intéresse plus particulièrement aux systèmes critiques. En effet, leurs dysfonctionnements peuvent avoir des répercussions sur le plan économique, environnemental ou humain (systèmes de contrôle dans les avions, les centrales nucléaires, ...). Dans de tels systèmes, où la part du contrôle logiciel est de plus en plus importante par rapport aux aspects mécaniques, il est important de pouvoir vérifier la sûreté aussi bien que la conformité aux spécifications. C'est l'objet des différentes techniques de vérification qui ont été développées ces vingt dernières années. Les méthodes formelles se sont vite imposées. Elles utilisent une formulation rigoureuse des besoins (spécifications) et une modélisation adéquate du fonctionnement du système, et permettent ainsi une validation rigoureuse des systèmes embarqués.

Bien que le problème de conformité d'un système à sa spécification soit en général indécidable, diverses méthodes de vérification de systèmes complexes ont été proposées, la preuve ([COQ]) et [Isa]), le test, le model-checking ([QS82],[BCM+90],[HNSY92]), et les méthodes d'interprétation abstraites car les méthodes de model checking se révèlent vite limitées. En effet, lorsque l'on s'intéresse à des propriétés numériques de programmes, par exemple « la valeur de cette variable entière ne dépasse jamais 100 », les méthodes de model-checking montrent leurs limites, parce que les espaces d'états engendrés sont trop gros ou infinis, et il faut alors manipuler des ensembles infinis de valeurs numériques. Dans ce cas, le problème peut être abordé principalement de deux façons :

- Certaines classes de problèmes/systèmes ont été identifiées comme décidables. L'inconvénient est que les langages d'expression des propriétés et/ou de modélisation des systèmes sont peu expressifs, et on ne peut vérifier que des propriétés concernant des systèmes « simples » . On peut aussi se ramener à ces classes en utilisant des abstractions, mais ces abstractions peuvent se révéler insuffisantes, et surtout difficiles à réaliser.
- Face à un problème intrinsèquement indécidable (ou simplement trop coûteux à résoudre), on peut se contenter d'une vérification *conservative* : seule une réponse positive est significative, c'est le cas en particulier des méthodes de model-checking symbolique, et des méthodes d'Interprétation Abstraite ([CC77]).

Nous nous plaçons donc dans le cadre de la vérification automatique de programmes qui comportent des variables numériques, et pour des propriétés numériques, par les méthodes d'interprétation abstraite. J'ai étudié les deux questions suivantes :

- La question de la *spécification*, c'est-à-dire l'expression des propriétés que l'on cherche à vérifier, est étudiée dans mes travaux de DEA. Ces travaux proposent une traduction d'une logique d'intervalles vers des automates observateurs de programmes synchrones ([HLR92]), qui permettent de vérifier des propriétés de façon plus simple, et d'utiliser les méthodes et outils existants.
- La question de la *vérification* de programmes numériques est étudiée dans ma thèse. Ces travaux cherchent à améliorer la précision et la performance d'une méthode existante pour permettre de vérifier des propriétés numériques sur des programmes de plus grande taille.

Ces travaux se sont effectués au sein de l'équipe synchrone, à Vérimag, et se sont donc placés dans le cadre de la programmation synchrone, que je vais présenter dans la sous-section suivante.

1.2 Cadre synchrone

Hypothèse synchrone La famille des langages et formalismes synchrones, parmi lesquels on peut citer Esterel ([BC85]), Signal ([GBBG86]) et Lustre ([HCRP91], ainsi que le formalisme StateCharts [Har84]), a été inventée et développée depuis le milieu des années 1980 pour résoudre le problème délicat de programmation des systèmes réactifs. L'hypothèse synchrone suppose que le temps de réponse d'un

système est nul. Cela signifie que le flot des sorties d'un programme est directement produit (sans délai) à partir du flot des données. Du point de vue de la programmation, l'hypothèse de synchronisme implique que le délai de communication entre plusieurs composants d'un programme est nul. En particulier, on peut composer des programmes sans changer le temps de réponse du système réel.

Le langage Lustre Le langage LUSTRE est un langage synchrone flot de données, dans lequel le calcul d'une nouvelle sortie dépend des entrées, et des entrées et sorties précédentes. La compilation de LUSTRE peut se faire vers un dialecte de C (ec) ou vers un langage d'automates (oc). Un certain nombre d'outils ont été réalisés autour du langage LUSTRE au sein du laboratoire Vérimag, comme on peut le voir sur la page [SYN].Dans la figure 2, nous nous restreignons aux outils de vérification.

La méthode utilisée pour vérifier la conformité d'un programme est celle des automates observateurs. Un automate observateur est un automate dont le but est d'observer le comportement des variables d'un programme et de signaler un comportement d'erreur. Dans le contexte synchrone, un automate observateur synchrone calcule à tout moment une fonction booléenne des variables du programme étudié (exprimé lui aussi dans le langage LUSTRE). Les outils LUSTRE réalisent ensuite un produit (synchrone) de l'automate du programme avec l'automate observateur, et vérifient que la valeur de la sortie booléenne `ok` est toujours `true` (voir la figure 1.2, dans lequel l'observateur "observe" la valeur des flots de données entrées et sorties du programme, et calcule la variable booléenne *ok* à tout instant).

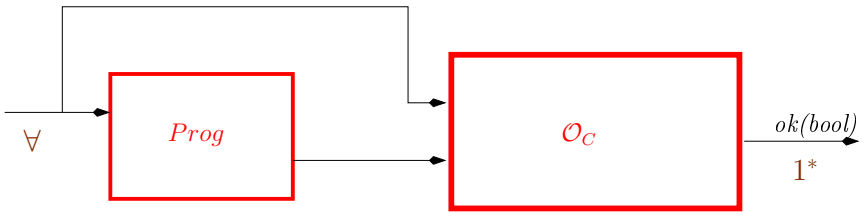

FIG. 1 – Observateur LUSTRE

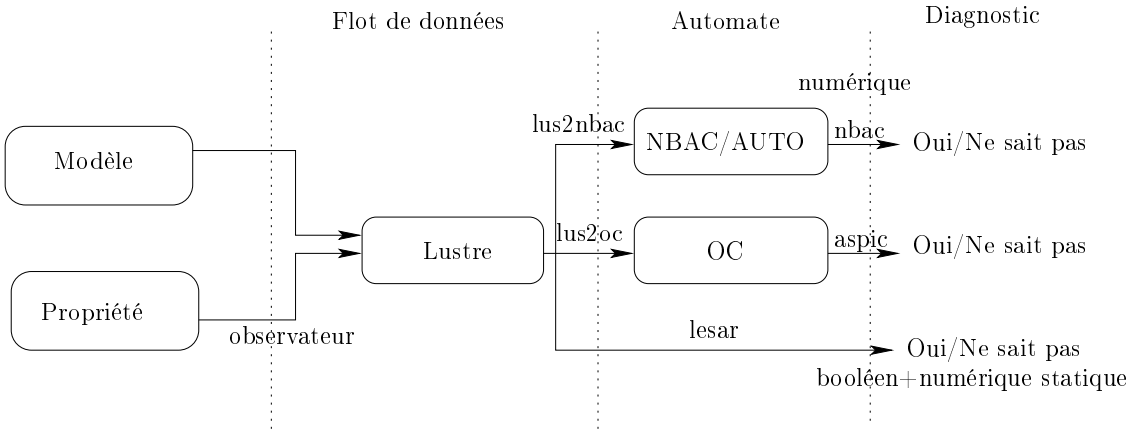

FIG. 2 – Outils LUSTRE de vérification

L'outil LESAR ([HLR92]) réalise le model-checking booléen d'un automate Lustre. L'outil NBAC est un analyseur à base d'Analyse des Relations Linéaires (voir la section 1.4) et de partitionnement dynamique [Jea00], qui permet de diriger l'analyse selon la propriété à prouver. Cet analyseur permet de prouver des propriétés de sûreté (« quelque chose de mauvais n'arrivera jamais »), mais pas de les infirmer (en revanche, l'analyse termine toujours).

Le format intermédiaire OC, qui est commun aux compilations de LUSTRE et ESTEREL ([PS87]), est un format d'automate avec ou sans variables booléennes. C'est le langage intermédiaire que nous choisiront pour connecter notre prototype ASPIC à LUSTRE.

1.3 DEA : du calcul des durées aux automates symboliques

Mes travaux de DEA ont eu pour objet la vérification automatique de programmes LUSTRE. Les propriétés que nous cherchons à vérifier sont décrites dans une logique temporelle particulière, QDDC ([Pan01]). Voici un exemple de formule de cette logique, qui est une logique d'intervalles :
« Pour chaque intervalle de taille 6 (*i.e.* pour chaque séquence de 7 tics consécutifs), on peut découper l'intervalle en deux parties : dans la première partie le nombre d'occurrences du signal q (*i.e.* le nombre de fois où la variable booléenne q vaut `true`) est inférieur à 2, dans le deuxième sous-intervalle, le nombre de signaux p à vrai est supérieur à 4 ». Il est important de noter ici que la taille des intervalles peut être non bornée, et que les constantes numériques peuvent être des paramètres.

Le but de ces travaux de DEA est de reconnaître des modèles de formules exprimées dans une logique avec un large pouvoir d'expression, à l'aide d'automates symboliques observateurs construits automatiquement, et qui peuvent éventuellement avoir un espace infini d'états. L'idée est d'utiliser le langage LUSTRE comme langage descriptif de programmes et de propriétés (via des observateurs générés à partir des formules à vérifier), et ensuite d'utiliser les outils de vérification qui gravitent autour de ce langage.

Nous avons identifié deux fragments de la logique QDDC qui se traduisent automatiquement vers des observateurs LUSTRE. La traduction de ces deux fragments a été implémentée et utilisée dans quelques études de cas. Cependant, certains exemples étudiés montrent les limites de NBAC, l'analyseur utilisé. Ceci nous amène donc à nous poser la question de l'amélioration de la précision dans les outils d'analyse des relations linéaires, qui est la méthode utilisée dans NBAC.

Il est intéressant de noter que ce travail permet non seulement la vérification de programmes, mais aussi le test de certaines propriétés QDDC sur des programmes. En effet, nous avons identifié un fragment déterministe de QDDC que nous savons transformer en automate observateur. Cet automate, compilé en même temps que le programme, permet de réaliser le test de la propriété à l'exécution.

Publication Ce travail a fait l'objet d'une publication à SLAP (Synchronous Langages, Applications and Programming, [GHR04]).

1.4 Doctorat : Accélération Abstraite pour l'amélioration de la précision en Analyse des Relations Linéaires

Le cadre d'étude de ma thèse est la vérification de propriétés numériques sur des automates à compteurs, ou *automates interprétés*. Ces automates interprétés peuvent être obtenus à partir de la compilation de programmes synchrones (avec ou sans observateur), ou bien de programmes classiques.

Nous cherchons à caractériser les propriétés du contrôle en associant à chaque point de contrôle de ces automates un invariant, c'est à dire une formule logique qui décrit la relation entre les variables *chaque fois que l'exécution du programme atteint ce point de contrôle*. Les propriétés non linéaires étant considérablement plus difficiles à traiter, nous nous restreignons à des propriétés qui sont des conjonctions de contraintes affines, c'est-à-dire de la forme $x + 2y \leq 6$, et à des fonctions de transition qui ont des gardes g_i et des actions affines a_j.

Si l'on note $\mathcal{A}_k$ l'ensemble des valuations des variables au point de contrôle k (c'est-à-dire l'ensemble des valeurs que peuvent prendre les vecteurs des variables), et que l'on observe le sous-graphe de la figure 3, alors on peut déduire l'équation :

$$\mathcal{A}_k = a_1(\mathcal{A}_{k_1} \cap g_1) \cup a_2(\mathcal{A}_{k_2} \cap g_2) \cup a_3(\mathcal{A}_k \cap g_3)$$

L'ensemble de ces équations pour tous les états de l'automate est un système de la forme $X = F(X)$. Pour un tel système, les théorèmes classiques de point fixe (Kleene) nous assurent l'existence du plus petit point fixe (lfp) donné par l'expression $\text{lfp}(F) = \bigcup_{n \geq 0} F^n(\emptyset)$.

Résolution du système L'ensemble cherché est donc la limite $F^*(X_0)$ d'une suite $X_0, ...X_n, ...$ d'ensembles, avec $X_{n+1} = F(X_n)$, où F est une fonction croissante. Cependant, ce calcul pose le problème de la représentation des ensembles d'états X_n, et celui de la convergence de la suite, qui est en général infinie.

Deux approches ont été étudiées précédemment : des résultats récents ont été obtenus par les méthodes d'*accélération* : selon la forme de la fonction F, on peut calculer directement F^* ([BW94], [CJ98], [FS00], [Bar05], cependant ces travaux ne s'appliquent qu'à des classes restreintes de programme. La méthode d'Analyse des Relations Linéaires (instance du cadre général de l'interprétation abstraite, sur le treillis des polyèdres, [Hal79]) s'attache quant à elle à calculer une approximation supérieure de la limite $F^*(X_0)$. L'application d'un opérateur d'*élargissement* permet d'extrapoler à partir des premiers termes de la séquence cette approximation supérieure. L'outil NBAC précédemment cité utilise entre autres cette méthode.

Cependant, les diverses études de cas (dont certains exemples de mon DEA), montrent que la méthode d'analyse des relations linéaires (à l'aide du treillis des polyèdres) souffre du manque de précision de la limite calculée, et que les travaux d'amélioration de la précision sont souvent *ad hoc*. Parmi ceux-ci, on peut citer des travaux d'amélioration de l'opérateur d'élargissement ([BHRZ03], de la séquence d'itération ([Hal93], [HPR97], [GR06]). Le gain de précision n'est toutefois pas satisfaisant. L'objectif de ma thèse est de trouver comment la méthode d'analyse des relations linéaires peut être améliorée par des techniques d'accélération.

Contribution Mon travail de thèse combine les méthodes d'accélération récentes, avec les méthodes classiques d'interprétation abstraite avec les polyèdres. Cette approche a deux principaux avantages :

- elle permet de traiter une classe large de programmes, la terminaison étant assurée ;
- elle permet d'améliorer la précision dans le cas de certaines boucles du programme, et donc la précision globale de l'analyse.

Dans cette thèse :

- Nous avons étudié les techniques existantes d'amélioration de l'analyse des Relations Linéaires et montré que ces différentes techniques ne sont que partielles et ne résolvent pas complètement le problème de manque de précision. Ces techniques sont diverses : la modification de l'opérateur d'élargissement, des heuristiques de parcours, de la stratégie de calcul.
- Nous avons étudié les techniques existantes d'accélération. Cet état de l'art conséquent a permis d'identifier des classes de transitions intéressantes en terme d'amélioration de la précision. Nous avons également étudié les différentes techniques de mise en œuvre dans les outils d'analyse par accélération.
- Nous avons défini la notion d'*accélération abstraite* de transitions affines gardées et dans divers cas particuliers (translations, ...), nous fournissons des algorithmes de calcul de surapproximation de l'enveloppe convexe de l'image d'un polyèdre par une ou plusieurs boucles. Dans certains cas, ces algorithmes donnent des résultats exacts, dans d'autres les résultats sont approchés mais plus précis que les invariants produits par les techniques d'élargissement.
- Ces techniques d'accélération approchées de boucles sont combinées à l'approche générale d'Analyse des Relations Linéaires, qui analyse des systèmes dont les fonctions de transfert ainsi que les graphes de contrôle sont quelconques. Nous introduisons des heuristiques portant sur le choix des boucles à accélérer, l'alternance des techniques d'accélération et d'élargissement ; dans certains cas nous proposons de modifier la structure du système afin de simplifier l'analyse.
- L'intérêt de cette approche est que par construction le gain de précision se fait sans perte d'efficacité. Pour valider l'approche, nous avons en parallèle de nos travaux théoriques développé un outil de vérification, ASPIC. Les expérimentations montrent une baisse du nombre d'itérations de l'analyse (donc une analyse plus efficace, même si certains point peuvent encore être améliorés) et les résultats des analyses sont plus précis.

Le prototype d'analyseur réalisé (que nous avons intégré à la chaîne de vérification de LUSTRE) peut être trouvé à l'adresse :

`http://laure.gonnord.org/pro/aspic/aspic.html`

Ce prototype est écrit en OCaml et comporte 20000 lignes de code (dont 5000 préexistantes).

Les expérimentations ont été menées d'une part sur de petits exemples « jouet », provenant de la communauté de la vérification, et d'autre part sur des exemples provenant de recherches d'autres doctorants du laboratoire. Nous avons par exemple effectué des analyses d'accessibilité sur des automates modélisant des comportements de programmes SystemC ([CMMC07]), et calculé des surapproximations de consommation d'énergie sur des automates provenant de modélisation de capteurs ([SMMM06]).

Publications Au début de ma thèse, j'ai participé à des travaux en cours sur l'amélioration du treillis des polyèdres, et j'ai cosigné un article de journal sur ce sujet (Formal Methods in System Design, [HMG06]). Par ailleurs, mes travaux de thèse ont fait l'objet d'une publication à SAS (Symposium on Static Analysis, [GH06]), et un article de journal est en cours d'écriture en vue de soumission à Formal Methods in System Design.

1.5 Etudes postdoctorales 2007-2008 : Étude de l'architecture Qinna, Qualité de Service.

Ce postdoc s'est inscrit dans le cadre du projet REVE (save Reuse of Embedded components in heteregeneous enVironmEnts). Le contexte de l'étude est toujours celui des systèmes embarqués, donc un contexte de ressources limitées, mais les applications sont différentes. Nous avons étudié ici des applications multimédia sur des supports embarqués, ces applications ne sont donc plus véritablement *critiques*. Cependant, on veut pouvoir parler des composants logiciels en terme de ressources, et de services fournis. Ces propriétés quantitatives sur les ressources font partie de la notion plus vaste de Qualité de Service fournie par un composant (ce que [BJPW99] appelle les contrats de niveau 4). Par exemple, suivant la qualité de la ressource réseau, l'affichage d'une image distante sera « bonne » ou « mauvaise ».

Une architecture logicielle à base de composants, Qinna, a été proposée dans [Tou05]. Cette architecture est une série de composants à ajouter à un logiciel afin de pouvoir gérer la qualité de service de chacun de ses composants. Plus précisément, Qinna donne un cadre général qui permet de :

- coder différents modes de fonctionnement des composants pour un même service à fournir.
- adapter les modes de fonctionnement des composants les uns par rapport aux autres : par exemple, si l'on veut une image de qualité supérieure, il faut demander un peu plus de temps à la ressource CPU.
- gérer l'adaptation des modes de fonctionnement selon des politiques d'adaptation données par le concepteur ou fournies par Qinna.

Dans ce contexte, notre travail a été formaliser les caractéristiques de qualité de service que traite Qinna, à simplifier et implémenter le tout, ainsi que des méthodes d'évaluation des performances du logiciel (monitoring, et preuve de propriétés de qualité de service : en terme de délai par exemple). Ce travail a permis de mettre en évidence

Expression et maintenance des contraintes de ressource dans Qinna L'étude de l'architecture Qinna a fait apparaître des mécanismes de maintenance de contraintes de ressources qui ne sont en fait pas des contraintes de liaison. Dans un premier temps, notre travail a donc consisté en une formalisation des différents types de contraintes (contraintes de comportement et contraintes de liaison entre composants), et aussi des mécanismes de maintenance et de contractualisation.

Cette formalisation a eu des conséquences sur l'architecture, elle a en effet en particulier permis :

- l'extension de Qinna au cas des composants qui fournissent de multiples services et qui peuvent avoir plusieurs instances ;
- l'extension de l'expressivité des contraintes dans Qinna à des contraintes de ressources intrinsèques aux composants ou à certains groupements de composants. Ces contraintes pourront éventuellement être exprimées en QML ([FK98]), un langage existant de descriptions de propriétés.

Publications La description de la mise en œuvre de Qinna sur un cas d'étude a été publié dans l'article **Resource usage formalization for Component-Based embedded Systems**, présenté en Octobre 2008 à *Real Time Systems, Wisla, Pologne*
La formalisation en terme de contraintes de ressources a été soumis à AICSSA 2009.

1.6 Etudes Postdoctorales 2008-2009

Je suis actuellement au LIP dans l'équipe Compsys. Mes travaux actuels visent à utiliser mon outil ASPIC à partir d'un code « full C ». Pour cela j'utilise l'outil ROSE qui propose des librairies de parsing et de génération de code pour le langage C :

`http://rosecompiler.org/`

ainsi que des méthodes de slicing.

Ce travail s'effectue afin d'utiliser ASPIC pour la détection du nombre de tours maximal que peut prendre une boucle. Ce travail servira ensuite à la synthèse de haut niveau de programmes destinés à être gravés sur puce.

Références

[Bar05] S. Bardin. *Vers un model checking avec accélération plate de systèmes hétérogènes.* Thesis, Laboratoire Spécification et Vérification, ENS Cachan, France, October 2005.

[BC85] Gérard Berry and Laurent Cosserat. The esterel synchronous programming language and its mathematical semantics. In *Seminar on Concurrency, Carnegie-Mellon University*, pages 389–448, London, UK, 1985. Springer-Verlag.

[BCM+90] J. Burch, E. Clarke, K. Mcmillan, D. Dill, and L. Hwang. Symbolic model checking : 10^{20} states and beyond. In *Proceedings of the Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 1–33, Washington, D.C., 1990. IEEE Computer Society Press.

[BHRZ03] R. Bagnara, P. M. Hill, E. Ricci, and E. Zaffanella. Precise widening operators for convex polyhedra. In R. Cousot, editor, *Static Analysis : Proceedings of the 10th International Symposium*, volume 2694 of *Lecture Notes in Computer Science*, pages 337–354, San Diego, California, USA, 2003. Springer-Verlag, Berlin.

[BJPW99] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins. Making components contract aware. *IEEE Computer*, 13(7), July 1999.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *CAV'94*, Stanford (Ca.), 1994. LNCS 818, Springer Verlag.

[CC77] P. Cousot and R. Cousot. Abstract interpretation : a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th ACM Symposium on Principles of Programming Languages, POPL'77*, Los Angeles, January 1977.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and Presburger arithmetic. In *CAV'98*, Vancouver (B.C.), 1998. LNCS 1427, Springer Verlag.

[CMMC07] J. Cornet, F. Maraninchi, and L. Maillet-Contoz. Formalizing systemc transaction-level models of systems-on-chip : a component-based approach. Unpublished, 2007.

[COQ] COQ. `http://coq.inria.fr/coq-fra.html`.

[FK98] Svend Frølund and Jari Koistinen. Qml : A language for quality of service specification. Technical report, HPL-98-10, 1998.

[FS00] A. Finkel and G. Sutre. An algorithm constructing the semilinear post* for 2-dim reset/transfer vass. In *25th Int. Symp. Math. Found. Comp. Sci. (MFCS'2000)*, Bratislava, Slovakia, August 2000. LNCS 1893, Springer Verlag.

[GBBG86] P. Le Guernic, A. Benveniste, P. Bournai, and T. Gautier. SIGNAL, a data flow oriented language for signal processing. *IEEE-ASSP*, 34(2) :362–374, 1986.

[GH06] L. Gonnord and N. Halbwachs. Combining widening and acceleration in linear relation analysis. In *13th International Static Analysis Symposium, SAS'06*, Seoul, Korea, August 2006.

[GHR04] L. Gonnord, N. Halbwachs, and P. Raymond. From discrete duration calculus to symbolic automata. In *3rd International Workshop on Synchronous Languages, Applications, and Programs, SLAP'04*, Barcelona, Spain, March 2004.

[GR06] D. Gopan and T. Reps. Lookahead widening. In *CAV'06*, Seattle, 2006.

[Hal79] N. Halbwachs. Détermination automatique de relations linéaires vérifiées par les variables d'un programme. Thèse de troisième cycle, University of Grenoble, March 1979.

[Hal93] N. Halbwachs. Delay analysis in synchronous programs. In *Fifth Conference on Computer-Aided Verification, CAV'93*, Elounda (Greece), July 1993. LNCS 697, Springer Verlag.

[Har84] D. Harel. Statecharts : A visual approach to complex systems. In *Advanced NATO Institute on Logics and Models for Verification and Specification of Concurrent Systems*, La Colle-sur-Loup, 1984.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous dataflow programming language LUSTRE. *Proceedings of the IEEE*, 79(9) :1305–1320, September 1991.

[HLR92] N. Halbwachs, F. Lagnier, and C. Ratel. Programming and verifying real-time systems by means of the synchronous data-flow programming language LUSTRE. *IEEE Transactions on Software Engineering, Special Issue on the Specification and Analysis of Real-Time Systems*, pages 785–793, September 1992.

[HMG06] N. Halbwachs, D. Merchat, and L. Gonnord. Some ways to reduce the space dimension in polyhedra computations. *Formal Methods in System Design*, 29 :79–95, 2006.

[HNSY92] T. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model-checking for real-time systems. In *LICS'92*, June 1992.

[HPR97] N. Halbwachs, Y.E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2) :157–185, August 1997.

[Isa] Isabelle. `http://isabelle.in.tum.de/`.

[Jea00] B. Jeannet. Partitionnement dynamique dans l'analyse de relations linéaires et application à la vérification de programmes synchrones. Thèse, Institut National Polytechnique, Grenoble, September 2000.

[Pan01] P.K. Pandya. Specifying and deciding quantified discrete-time duration calculus formulae using dcvalid. In *Real-Time Tools, RTTOOLS'2001*, Aalborg, August 2001.

[PS87] J. A. Plaice and J-B. Saint. The LUSTRE-ESTEREL portable format. Unpublished report, INRIA, Sophia Antipolis, 1987.

[QS82] J-P. Queille and J. Sifakis. Specification and verification of concurrent systems. In *in CESAR International Symposium on Programming Proceedings*, 1982.

[SMMM06] L. Samper, F. Maraninchi, L. Mounier, and L. Mandel. GLONEMO : Global and accurate formal models for the analysis of ad-hoc sensor networks. In *Proceedings of the First International Conference on Integrated Internet Ad hoc and Sensor Networks (InterSense'06)*, Nice, France, May 2006.

[SYN] SYNCHONOUS Team. `http://www-verimag.imag.fr/SYNCHRONE/`.

[Tou05] J-C. Tournier. *Qinna : une architecture à base de composants pour la gestion de la qualité de service dans les systèmes embarqués mobiles.* PhD thesis, INSA-Lyon, 2005.

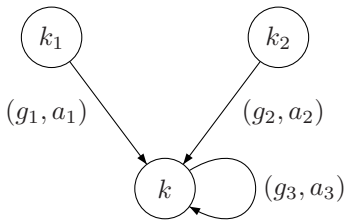

FIG. 3 – Une partie du graphe de flot de contrôle